TECHNISCHE UNIVERSITÄT DRESDEN

Faculty Computer Science Chair Software Technology, Software Architecture of Complex Systems

Master Thesis

Mitigating Emergent Safety and Security Incidents of CPS by a Protective Shell

Leonard Wagner

Matriculation number: 4767780
Matriculation year: 2018

4th September 2023

Supervisor
Dr.-Ing. Sebastian Götz

Supervising professor
Prof. Dr. Frank J. Furrer

Acknowledgements

First and foremost, I wish to express my deepest gratitude to Prof. Dr. Frank J. Furrer for his continuous support and insightful advice throughout this research journey. His meticulously crafted topic proposal, coupled with his profound interest in this study field and invaluable literature recommendations, have played a central role in shaping this thesis. His collaboration, alongside with Dr.-Ing. Sebastian Götz, consistently provided highly productive consultations, guiding me through the complexities of my research.

I would also like to express my sincere appreciation to Dr.-Ing. Sebastian Götz. His extensive experience, innovative ideas, and guidance on crucial aspects have been instrumental in bringing clarity and depth to my work. Additionally, his assistance in navigating through organisational matters has been immensely beneficial.

On a personal note, I owe a heartfelt thanks to my girlfriend. Her unwavering belief in my abilities and her relentless support, have been indispensable and a constant source of strength. Her thoughtful gestures, have lightened the load during challenging periods while she ensured that I remained focused on this thesis.

Lastly, I would also like to convey my warm gratitude to my family and friends for their unyielding support and encouragement, while spurring me onward even when faced with obstacles.

In conclusion, while this master thesis bears my name, it would not have been possible without the contributions of these remarkable individuals with their support, guidance and belief in me. For this, I am truly grateful.

Master Thesis Topic for Leonard Wagner

"*Mitigating Emergent Safety and Security Incidents of CPS by a Protective Shell*"

Sommersemester 2023

Betreuer: Prof. Dr. Frank J. Furrer
Zweitbetreuer: Dr. Sebastian Götz

Context

In his *Hauptseminar* paper: "[Wiper]NotPetya - Eine detaillierte Designanalyse des teuersten Cybersicherheit-Weckrufs"[1], Leonard Wagner worked on cybersecurity. He expressed his interest to continue the work on safety and security of cyber-physical systems.

Safety-Critical Cyber-Physical Systems

Today's world heavily relies on *cyber-physical systems* ([Alur15]). Many of these systems are *safety-critical*, i.e., a failure or malfunction may lead to damage or even loss of life. Good examples are *automated planes* ([Hersch20]) or *autonomous vehicles* ([Liu18]).

Designing, implementing, and operating *trustworthy cyber-physical systems* have become a vital engineering discipline (e.g.: [Heegaard15] [NAP12] [BMW20] [Romanovsky17]). Many methodologies, technologies, and development processes exist (e.g.: [Kavallieratos20]).

Protective Shell

Unfortunately, the *complexity* of modern cyber-physical systems (CPS) is enormous and continuously rising. In fact, most CPS are *system-of-systems*, i.e., networked, interoperating, dependent systems (e.g.: [Haimes19]).

Today, the awareness slowly rises that it is becoming more challenging to ensure such complex systems' trustworthiness via architecting, design, realization,

[1] Leonard Wagner, Hauptseminar (Prof. Dr. Frank J. Furrer) «Engineering Principles for Safety and Security in Cyber-Physical Systems», SS 2022, July 2022.

maintenance, and operations. Too many failure modes, malfunctions, malicious attacks, and emergent behaviour exist.

Therefore, new ways of improving the trustworthiness of CPS's are being investigated. One promising avenue is the *runtime monitoring* of the CPS (e.g.: [Brukman11]): The CPS is continuously monitored, and anomalous or dangerous behaviour is detected and contained. A well-designed *protective shell* around the cyber-part of the CPS then can act as a "Last Defence" and protect the CPS from failures and attacks or from their negative impacts (Figure 1).

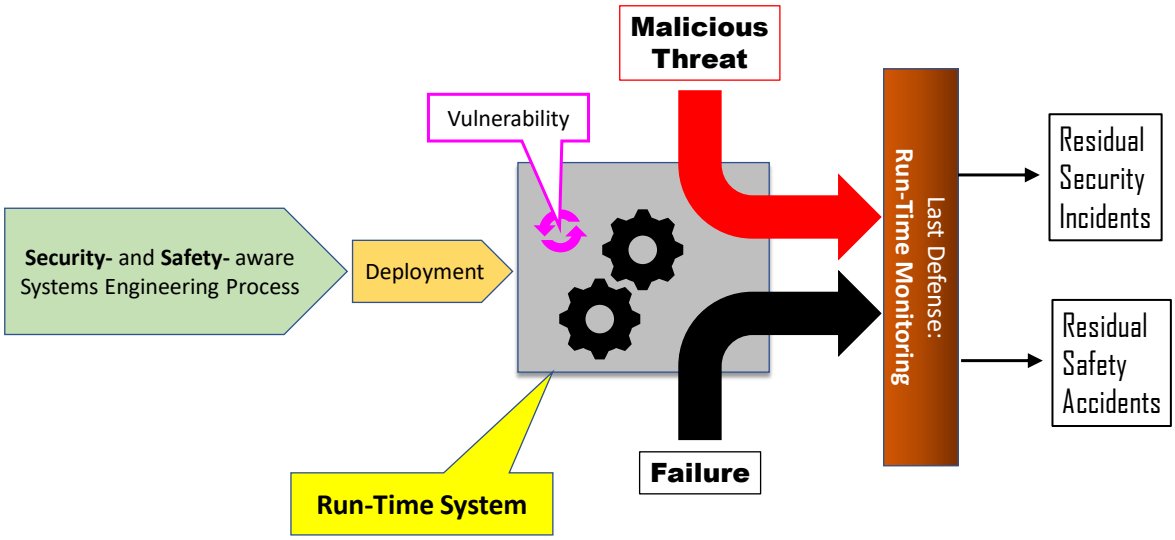


Figure 1:

The idea of a protective shell was presumably first presented by Lance Eliot ([Eliot16]): He proposed it under the name of «AI Guardian Angel Bot» to improve the trustworthiness of machine-learning systems. Here the more technically-oriented (and less esoteric) term «Protective Shell» is used (Definition 1).

Definition 1: "Protective Shell":

Independent hardware/software layer overlaid on a system using additional information, such as supplementary sensor data or log files, to detect, analyze, interpret, and mitigate potentially dangerous or unsafe system behavior, either in realtime, near-realtime, or batch processing.

The protective shell (Figure 2) works with many inputs, such as:

- Signals from the application software;
- Operational data, such as log, exception, and error files;
- Operational data, such as packet analyzers, anti-malware programs;
- Additional sensor data from the physical part.

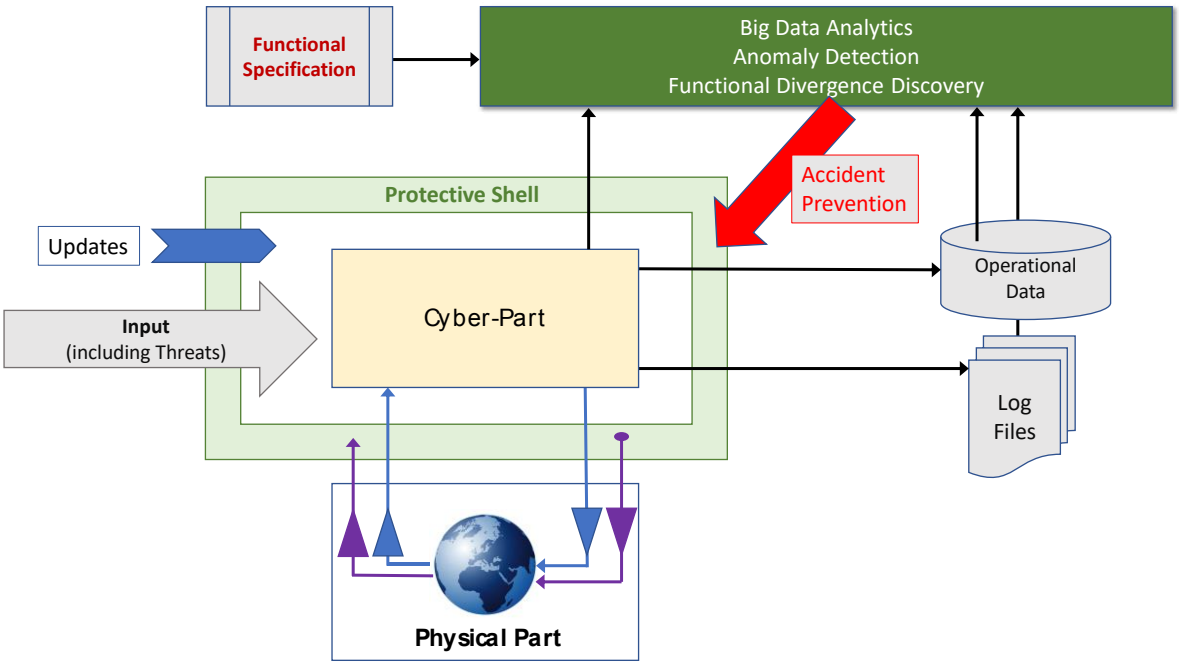


Figure 2: Protective Shell for a CPS

The protective shell uses powerful real-time or near-real-time *analysis techniques* to identify anomalous or dangerous behaviour or diverging behaviour from the functional specification. Whenever a dangerous or unsafe behaviour is detected (or suspected), the protective shell immediately takes corrective, protective action, such as leading the system into a safe state, go into degraded operation, or hand over to manual operation.

Research Methodology

Many research methodologies for this type of research exist. A particularly well suited methodology is the "design science". Central point of the design science are the *research questions* (Definition 2).

Definition 2: "Research Question":

A research question is a question that a study or research project aims to answer. This question addresses an issue or a problem, which, through analysis and interpretation of data, is answered in the study's result

https://research.com/research/how-to-write-a-research-question

The research questions are formulated after initial research. They constitute a strong guidance for the thesis work. Good research questions, therefore, a crucial for the efficient and effective work ([RQ21]).

Work Plan

The work plan is as follows:

1. Literature research and study, both theoretical foundations and relevant material;
2. Formulate the research question
3. Select 4…5 well-documented *safety accidents* and 4…5 well-documented *security incidents* with substantial damage from the literature/Web;
4. Analyse and document the accidents and incidents;
5. Determine and analyse the causes of the accidents/incidents and identify the weaknesses of the CPS which enabled the accidents/incidents;
6. Develop specific *protective shells* for the individual accidents and incidents;
7. Assess the impact of the protective shells on each incident/accident: How effective would the protection shell have been?
8. A protective shell requires additional hardware and software and thus increases the complexity of the system. For each of the specific protective shells: Have new failure modes been generated or has the attack surface been enlarged? Are additional protection measures required?
9. Use the experience gained from the development of the specific protective shells to derive and formulate general *architecture and design principles* for protective shells;
10. Evaluate and assess the value of the “Protective Shell”-technology as a general means to reduce risks in safety-critical or security-critical CPS;
11. Assessment: Have the research questions been answered?
12. Present conclusions and suggestions for future work.

Deliverables

Master Thesis (Format according to TUD standards) and Presentation (Verteidigung).

Prof. Dr. Frank J. Furrer

Dresden, April 19, 2023

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty Computer Science Chair Software Technology, Software Architecture of Complex Systems

Abstract

In today's modern world, *Cyber-Physical Systems* (CPS) have gained widespread prevalence, offering tremendous benefits while also increasing society's dependence on them. Given the direct interaction of CPS with the physical environment, their malfunction or compromise can pose significant risks to human life, property, and the environment. However, as the complexity of CPS rises due to heightened expectations and expanded functional requirements, ensuring their trustworthy operation solely during the development process becomes increasingly challenging.

This thesis introduces and delves into the novel concept of the "*Protective Shell*" – a real-time safeguard designed to actively monitor CPS during their operational phases. The protective shell serves as a last line of defence, designed to detect abnormal behaviour, conduct thorough analyses, and initiate countermeasures promptly, thereby mitigating unforeseen risks in real-time.

The primary objective of this research is to enhance the overall safety and security of CPS by refining, partly implementing, and evaluating the innovative protective shell concept. To provide context for collaborative systems working towards higher objectives — common within CPS as system-of-systems (SoS) — the thesis introduces the "*Emergence Matrix*". This matrix categorises outcomes of such collaboration into four quadrants based on their anticipated nature and desirability. Particularly concerning are outcomes that are both unexpected and undesirable, which frequently serve as the root cause of safety incidents and security breaches in CPS scenarios. The protective shell plays a critical role in mitigating these unfavourable outcomes, as conventional vulnerability elimination procedures during the CPS design phase prove insufficient due to their inability to proactively anticipate and address these unforeseen situations.

Employing the *design science research methodology*, the thesis is structured around its iterative cycles and the research questions imposed, offering a systematic exploration of the topic. A detailed analysis of various safety accidents and security incidents involving CPS was conducted to retrieve vulnerabilities that led to dangerous outcomes. By developing specific protective shells for each affected CPS and assessing their effectiveness during these hazardous scenarios, a generic core for the protective shell concept could be retrieved, indicating general characteristics and its overall applicability.

Furthermore, the research presents a *generic protective shell framework*, integrating advanced anomaly detection techniques rooted in *explainable artificial intelligence* (XAI) and *human machine teaming*. While the implementation of protective shells demonstrate substantial positive impacts in ensuring CPS safety and security, the thesis also articulates potential risks associated with their deployment that require careful consideration.

In conclusion, this thesis makes a significant contribution towards the safer and more secure integration of complex CPS into daily routines, critical infrastructures and other sectors by leveraging the capabilities of the generic protective shell framework.

Keywords

Protective Shell, Run-Time Monitoring, Cyber-Physical Systems, Safety Accidents, Security Incidents, Explainable Artificial Intelligence

Contents

List of Figures

List of Tables

1 Introduction

This thesis starts with a concise introduction addressing the contextual background of the research topic. It introduces the evolutionary trajectory of digital systems and the emergence of *Cyber-Physical Systems* (CPS). The significance of CPS within the modern digital era is underscored, emphasising their pervasive characteristic across various industry sectors.
Subsequently, the primary research problem confronting stakeholders in the CPS domain is articulated, alongside the expectations towards the research for addressing this problem by providing a suitable and reasoned solution. Moreover, the relevance of this issue will be discussed further, especially in the sphere of safety and security-critical CPS.
Additionally, the thesis states its embraced vision and mission together with its anticipated impact, broader goals with key objectives, and intended contributions within the CPS application domain.
Finally, the thesis will be outlined, presenting an overview of the upcoming chapters along with a condensed summary of their content, to offer insights into the selected structure of this thesis.

1.1 Background and Context

In the history of technological evolution, few developments have been as transformative as the rise of digital systems. From the rudimentary computers of the mid-20th century to the pervasive smart devices of today, digital systems have continually reshaped the interaction with the world. As the modern society transitioned into the 21st century, a new paradigm emerged at the intersection of the computational and physical realms: *Cyber-Physical Systems* (CPS).
CPS represents a harmonious merge of computational elements, such as software applications, algorithms, and communication networks, with physical processes that directly measure (through sensors) and interact (through actuators) with the real world. This symbiotic relationship allows CPS to not only gather data from the physical environment but also to influence it in real-time, creating a *feedback loop* of continuous interaction.
The significance of CPS in today's digital era cannot be overstated. They hold the promise of substantial economic and societal benefits across a multitude of domains, including aerospace, production automation, transportation, and healthcare. For instance, in the transportation domain, CPS can be used in the form of autonomous vehicles that seamlessly integrate the cyber- and physical-part of such systems to navigate complex urban environments. In healthcare, CPS finds application in advanced medical devices that monitor patient vitals and administer treatments based on real-time data.

However, the adaptability of CPS extends beyond mere utility. They are envisioned as solutions to some of the most pressing challenges of current times. As the population of industrialised countries struggles with issues like ageing demographics, resource limitations, and the shift towards sustainable energies, CPS offer innovative solutions. They have the potential to enhance efficiency and human comfort by reducing manual labour, which makes them indispensable in the pursuit for a better future.
Yet, with great potential comes great responsibility. The direct interaction of CPS with the physical world means that their malfunction or compromise can have direct and often detrimental effects on their operation environment and the entities (including humans) therein. This great drawback of using CPS will be elaborated in more detail in the subsequent section about the research problem.

1.2 Research Problem

In the modern world, the rapid growth of CPS utilisation across industries is undeniable. These systems have seamlessly integrated into various facets of daily lives, becoming pervasive and indispensable. Their contributions span a wide spectrum, as previously stated, from enhancing comfort to significantly reducing manual labour. The transformative potential of CPS is evident in their ability to revolutionise industries, streamline processes, and elevate the quality of life.
However, as with all technological advancements, the rise of CPS brings forth a set of challenges. The research problem addressed in this thesis revolves around *ensuring the safety and security of complex CPS*, thereby enhancing their *trustworthiness during operation*.
As expectations and demands from stakeholders in the CPS domain on these systems grow, so does their *complexity*. Projects expand in scale, involving more engineers, longer development times and encompassing higher functional requirements. This escalating complexity is further amplified when considering many of today's CPS as *cyber-physical systems of systems* (CPSoS). In a CPSoS, multiple constituent systems collaborate, forming a cooperative mesh to achieve objectives that would be unattainable by any individual system. While this collaboration enhances functionality, it also introduces a heightened level of intricacy, making the resulting system of systems (SoS) considerably more complex and the inner workings less understandable even to their engineers.
This higher complexity of CPS is the root cause for the identified challenges, which also raises safety and security concerns from experts, hinting to safety accidents and security incidents involving CPS occurring occasionally with sometimes catastrophic outcomes.
With ever so rising complexity, building safe and secure CPSoS slowly transitions to an unattainable task, even when adhering to industry standards and employing established engineering patterns for security and safety. The integration of multiple systems leads to unexpected emerging properties, some of which negatively impact the overall system and its trustworthy operation.
Vulnerabilities, whether they arise from oversights in the development process, through inadequate risk management, or gaps in testing and model verification, pose significant threats. Furthermore, the sheer volume of such CPS in operation means that even if the probability of a malfunction, failure or malicious exploitation resulting from an overlooked vulnerability or weakness in the system is low, the likelihood of at least a few of the mass-produced systems encountering extreme, unlikely operational events, such as a bit flip caused by radiation, becomes a substantial concern.
Another pressing issue is the limitation in testing complex CPS. It is virtually impossible to test these systems for every conceivable situation by minding every possible operation condition as there are too many, leading to a scenario where some testing phases can't be completed and are inadvertently shifted to the end-users, sometimes with catastrophic consequences.

Recognising those challenges, there's an essential need for effectively safeguarding complex CPS during operation. The purpose and objectives of this master thesis regarding this research problem will be presented in the following section.

1.3 Purpose and Objectives

The research problem underscores the necessity for innovative and effective strategies to ensure the reliable operation of complex CPS by enhancing their overall safety against failures and malfunctions, as well as their security against exploitations of vulnerabilities within the system. Furthermore, addressing negative emergent properties causing abnormal and unexpected behaviour, that have detrimental effects on the current operation state, is essential to mitigate a substantial portion of safety accidents and security incidents involving CPS.
The purpose of this master thesis is to investigate a novel concept - the "*Protective Shell*" - and evaluating its efficiency in addressing the research problem. The protective shell concept shifts the focus from attempting to reduce residual risks of a CPS to an acceptable level exclusively during the design phase, to additionally leveraging run-time mechanisms to mitigate still present, more concealed risks.
It has become evident that focusing exclusively on enhancing the already highly sophisticated development processes, utilising multiple effective vulnerability elimination procedures, falls short of achieving trustworthy operation for complex CPS. Therefore, extending safety and security measures beyond design-time and integrating dynamic safeguards during runtime is a promising strategy. This addition holds the potential to substantially contribute to the overarching objective of *ensuring the trustworthy operation of CPS* to a satisfying extent.
The concept should encompass mechanisms capable of promptly detecting unexpected behaviour, assessing the CPS's own ability to handle such anomalies, and determining whether corrective or protective actions are necessary. This real-time monitoring and intervention approach tries to enhance the overall safety and security of CPS by mitigating residual runtime risks, especially in scenarios where unlikely events occur, such as sensor or actuator failures or vulnerability exploitations.

The protective shell concept was initially proposed by Dr. Lance Eliot in his book "AI Guardian Angel Bots for Deep AI Trustworthiness" [1] and later refined by Professor Frank J. Furrer in his scientific article "Safe and secure system architectures for cyber-physical systems" [2]. Throughout this thesis, the protective shell concept will be thoroughly elaborated and further refined. Through a rigorous evaluation process, the viability of the protective shell effectively addressing the research problem will be assessed.
An essential aspect of this work involves enhancing the protective shell's ability to identify abnormal behaviours beyond comparing static functional specifications. This enhancement will incorporate *advanced anomaly detection* techniques, integrating artificial intelligence and machine learning, thereby improving the protective shell's effectiveness in timely and accurately detecting abnormal CPS behaviour. Additionally, the trained machine learning models can also be used to create suitable intervention actions to lead the CPS back to a safe and secure operation state.
This and other extensions contribute to the broader objective of expanding the protective shell's applicability across diverse domains, making it universally implementable for most CPS scenarios with just slight configurative adjustments. It entails providing engineers with the tools necessary to implement their own specific version of the protective shell, facilitated by a versatile and dynamically adjustable framework. The *generic framework* should not only encompass a foundational architecture template but also diverse plugins for different de-

tection mechanisms and interventions, which can be autonomously integrated as required. Additionally, a comprehensive set of *architecture and design principles* will provide guidance throughout the entire lifecycle of each specific protective shell, spanning design, implementation, maintenance, and operation.
Furthermore, an evaluation methodology will be provided, offering insights into how effective each individual protective shell is within its operational environment, validating its efficiency in safeguarding CPS and fulfilling its intended purpose.
In summary, the overarching objective of this research task is to propose an innovative solution that ensures the effective realisation of CPS benefits without compromising their security or safety. By refining the protective shell concept, enhancing its ability to identify abnormal behaviour and propose appropriate interventions, while providing a comprehensive generic framework, this research aims to promote the widespread adoption of runtime monitoring in CPS. Therefore, suggestions made in this thesis are well-founded and pragmatic while strategically aimed to increase *industry acceptance* for this novel concept. The subsequent sections will present the concise vision and mission of this thesis, summarising the most fundamental points.

1.3.1 Thesis Vision

The thesis envisions a modernised society empowered by the capabilities of complex and highly advanced CPS in a manner that effectively leverages their tremendous inherent benefits without compromising their trustworthy operation by lowering safety or security standards.
CPS(-oS) can become highly complex, making it increasingly difficult to fully understand their intrinsic behaviour, even for the engineers designing these systems. As a result, concealed vulnerabilities are more likely to evade the risk management processes during development, and unexpected emergent behaviour among system components can lead to safety accidents and security incidents. In order to mitigate these multifaceted risks, the thesis envisions a solution based on reliable and efficient run-time monitoring. This solution should utilise machine learning techniques to swiftly identify operation anomalies and initiate rectifying measures, leading the system back to a trustworthy operation state. This solution is envisioned to be a protective shell, serving as the last line of defence against threats and failures.
To achieve widespread acceptance, efficiency and applicability within the CPS industry, the protective shell necessitates a generic architecture, universal design principles, and add minimal complexity to the overall system.
Through this visionary approach, the thesis aims to strike a balance between innovation and safeguarding, enabling these complex CPS to deliver their advantages while operating with unwavering safety and security.

1.3.2 Thesis Mission

The primary mission of this master's thesis is to enhance safety and security of complex CPS by introducing an additional layer to the defence-in-depth strategy, embracing the runtime system - the protective shell. Therefore, a generic protective shell framework will be developed, partly evaluated and presented. This framework contains design and architecture principles as well as concepts for advanced anomaly detection, mitigation mechanisms and interventions. The goal is to aid industry acceptance and accelerate the adoption rate among different manufacturers.
Embedded within the thesis is a fundamental mission to raise awareness about the current challenges complex CPS are facing, as this marks the first crucial step, and to show a viable

solution addressing these issues. To demonstrate the effectiveness of the protective shell, it will be implemented in CPS that experienced or were involved in accidents and incidents. During the evaluation, it will be assessed whether the protective shell can completely prevent the hazardous situation or dramatically reduce its damage potential. Ultimately, the thesis has the objective of providing the previously mentioned generic protective shell framework. Above all, the effort put into this thesis should pave the way for wide-spread usage of monitoring and real-time intervention solutions across the CPS domain, thereby improving the trustworthy operation of its systems.

1.4 Thesis Outline and Structure

This thesis will maintain its focus on a theoretical exploration rather than delve into real-world practical implementations of the protective shell concept in specific CPS. Additionally, no specific code library for the generic protective shell framework will be provided. The primary concern will be on scientific literature reviews and evaluations of post-incident and post-accident reports regarding safety accidents and security incidents involving CPS.
This work is precisely structured to offer an in-depth elaboration of the protective shell concept, recognising its promising capabilities for complex CPS as a viable solution to address their inherent challenges towards trustworthy operation.

The introduction has set the stage, elucidating the context and the central research problem as well as the thesis' overarching purpose with its vision and mission. It outlines the expectations and specific objectives that underpin the subsequent chapters to depict the common thread leading through the thesis.
Following the introduction, the thesis delves into the methodological approach adopted for this research 2, in this case the *design science research*. It sheds light on the balance between the real-world relevance and the scientific stringency applied throughout the study. Therefore, the design science cycles get explained alongside with the imposed research questions, which in turn define the structure of the subsequent chapters by contributing to the individual cycles and addressing those questions.

After stating the formalities, the thesis explores the *state-of-the-art* by consulting the underlying scientific knowledge base to define basic terms and paradigms needed. By describing CPS in chapter 3 and the protective shell concept in chapter 4, it introduces the reader to the safety- and security-critical aspects of this domain, highlighting the challenges CPS face when only using conventional risk management procedures as well as the expectations towards the innovative protective shell concept, thus also towards this thesis. Within these two chapters essential groundwork is done presenting concepts like CPSoS and its emergence effects on the resulting system as well as dividing the system architecture into the design-time and run-time architecture and the notion of run-time monitoring. All in all, the two chapters contribute greatly in establishing a robust foundation upon which the thesis can confidently build its own research.

In the subsequent chapter regarding specific protective shells 5 the new concept will be implemented in CPS from various industry sectors that were involved in well-documented safety accidents or security incidents. It offers a tangible perspective, presenting in-depth case studies where the main vulnerabilities and weaknesses present in the CPS that led to such disastrous outcomes will get retrieved from post-accident/incident reports released by trusted sources. After identifying these vulnerabilities, each individual protective shell is configured with specialised mitigation mechanisms to efficiently detect and counteract the abnormal behaviour exhibited during operation. After that, evaluations are conducted ad-

dressing the question, if it would have effectively safeguarded the CPS when implemented beforehand, and to what extent. To sum up the findings and the lessons learned from those case studies, the similarities between all developed protective shells and their evaluation results are enumerated, hinting to a *generic core* that every protective shell should have, providing insights what characterises their inner workings.
These real-world scenarios anchor the theoretical concepts discussed in the chapters earlier, emphasising the practical challenges and solutions in implementing protective shells across diverse systems.

Following, the chapter on artificial intelligence (AI) 6 enables the perspective switch from a reactive view, developing protective shells utilising deterministic, known information from reports, to a proactive view, trying to remedy potential hazardous scenarios before they manifest during operation. There, the important characteristics of *machine learning models* are presented and how they can be employed in protective shells for *advanced anomaly detection* and intervention planning. It highlights essential considerations when integrating AI into this safety- and security-critical domain and presents some suitable ML models as well as training approaches.

Once the key terms, concepts, mechanisms and guidelines are presented in the previous chapters, the generic protective shell framework can be created in chapter 7. There, the state-of-the-art foundations are paired with the generic core found during the case studies and the suggestions made for ML models. Additionally, the suitable reference architecture *MAPE-K* is refined for better applicability to the concept of protective shells and adoptions are made to support *human machine teaming*. Furthermore, a *plugin catalogue* will get presented offering a wide variety of modules for detection mechanisms and intervention actions, alongside with *architecture and design principles* discovered throughout the thesis.

Collectively, these elements support engineers throughout the development and implementation of their specific protective shells by guiding them through the individual processes, presenting different configuration options for great suitability while also reducing the likelihood of design flaws. Lastly, an evaluation methodology is presented how specific protective shells can be assessed on their effectiveness.
The recurring theme throughout this thesis is the indispensable role of designing resilient, safe and secure CPS in an era of increasingly complex systems. The protective shell concept emerges as a promising extension to the conventional safeguarding mechanisms employed in CPS, offering advanced protection against concealed vulnerabilities and weaknesses that slipped through the risk management process. Through a harmonious combination of theoretical discussions and practical insights, the thesis draws a comprehensive picture of the current CPS challenges, opportunities and tries to pave the way towards increased trustworthy operation of CPS.

2 Design Science Research Methodology

This thesis follows the *design science research* (DSR) methodology as it seems to be a well-suited, problem oriented option for this research topic. This chapter will shortly introduce design science research as well as the research questions for the thesis. To dive deeper into DSR than this chapter, it is recommended to have a look at the "Design Research in Information Systems" by Hevner and Chatterjee [3].

The design science research methodology is a systematic approach that create and evaluate innovative solutions to real-world problems under scientific principles in form of so called *artefacts*.
Therefore, artefacts represent and embody the proposed solution by the researcher and is typically evaluated as well as refined throughout the research process. This methodology is especially common in the computer science domain. Typical artefact solutions include software prototypes, conceptual frameworks or algorithms.

2.1 Relevance-, Rigor- and Design Cycle

The DSR process, as depicted in figure 2.1, is divided into three cycles, which will be presented in the following paragraphs. They should be part of every DSR project but don't have to follow a sequential order, rather work in symbiosis to create the artefact. Nevertheless, the term cycle means that it can be refined further with each subsequent iteration.

Relevance Cycle Since the DSR methodology is problem oriented, the first step is to define the context or environment where the research problem occurs. That could be a branch of industry, a specific science field, or many more. However, by looking at this environment it should be clear which actors participate, which technical and organisational structures are in place and the existing opportunities as well as challenges actors and organisation face.
After that, the relevance cycle sets three important attributes for the artefact. The application context, the requirements and lastly the acceptance criteria. Those attributes will also be important for the artefact's evaluation later on.

Rigor Cycle The rigor cycle is responsible for distinguishing the DSR from industrial practice. All work steps in the relevance and design cycle are also part of the standard developing/engineering process in the industry.

The rigor cycle separates that by abstracting knowledge gained through the DSR process and drawing principles or theoretical conclusions.
The foundation of the rigor cycle is the *knowledge base*, which incorporates existing knowledge, state-of-the-art, theory, and related pre-existing artefacts. The purpose of retrieving and collecting knowledge in the knowledge base is to influence the design process.
Following the evaluation of the artefact, the acquired knowledge will be fed back and expand the knowledge base for future reference.

Design Cycle In the design cycle, the artefact finally gets planned and created. Following that, it gets evaluated according to the expectations and goals set in the relevance cycle. If the artefact successfully passes the evaluation, the DSR process is finished. Otherwise, the design cycle has to be absolved again until the acceptance criteria are fulfilled.

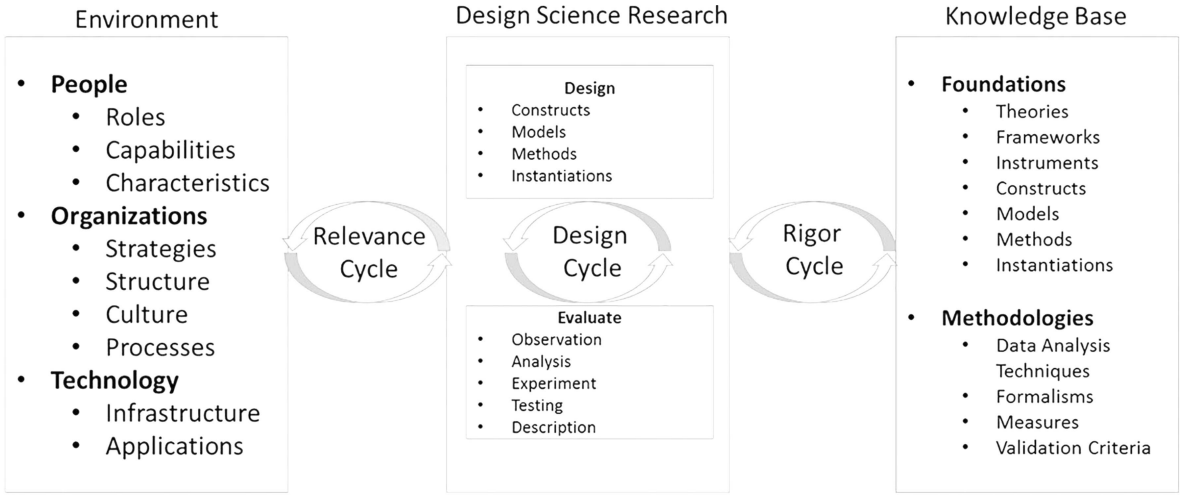


Figure 2.1: Design Science Research Framework (based on Hevner [3]) [4]

2.2 Research Questions

Research questions play a fundamental role in guiding the entire research process by specifying the scientific interest that the entire research effort is targeted at. They are the critical inquiries that define the scope, purpose, and direction of the study. These research questions serve four essential purposes:

Problem Identification Research questions help to identify the specific problems or challenges that the study aims to address. They highlight gaps in knowledge or areas where existing solutions are insufficient, prompting researchers to seek innovative approaches.

Goal Clarity Clear research questions articulate the goals and objectives of the scientific work, maintaining focus and preventing the study from becoming too broad or losing its direction.

Knowledge Advancement By seeking answers to well-defined research questions, design science research contributes to the advancement of knowledge in a particular field. The solutions developed and insights gained can be used as a basis for future research and improve understanding of the problem domain.

Rigorous Evaluation Research questions set the stage for evaluating the proposed design artefacts. They establish criteria for assessing the success or effectiveness of the proposed designs, ensuring that the research outcomes have practical significance.

The research questions for this thesis are as follows:

1. Can the trustworthiness of safety and security-critical cyber-physical systems be improved by implementing a protective shell?
2. Can a generic core for a protective shell be developed which is applicable to multiple application domains?
3. Is it possible to mitigate the risks that emerge when implementing a protective shell to reduce overall residual risks for safety and security?

The research questions chosen in this thesis focus on investigating the potential benefits of implementing a protective shell in safety and security-critical cyber-physical systems. Through addressing these questions, the thesis seeks to provide answers on whether the trustworthiness of such systems can be improved, even when considering potential risks associated with the implementation of a protective shell. Additionally, the study aims to identify a generic core for the protective shell concept that is not context-based but broadly applicable, making it suitable for adoption across various industries.
Ultimately, this research aims to contribute valuable insights to the field of CPS engineering and initiate future efforts in safety and security enhancement. Through its findings and conclusions, this thesis intends to contribute meaningfully to the advancement of safety and security practices in CPS.

3 Cyber-Physical Systems

This chapter presents a comprehensive and fundamental introduction to *cyber-physical systems* (CPS) without diving into excessive detail. The primary focus of this chapter lies in the critical aspects of safety and security associated with CPS. Therefore, after providing definitions and briefly explaining their architecture, as well as their significance in the modern, digitalised world, the risks related to CPS usage will be highlighted. To ensure that the benefits of CPS outweigh the risks, it is crucial to enhance their trustworthiness through rigorous and robust methodologies.
Subsequently, the *engineering principles for safety and security*, which are considered industry standards, will be introduced, along with a few concrete examples.
Finally, the concept of *system of systems* (SoS) will be explained, along with its impact on CPS. The main concern will be regarding the opportunities introduced by emergent behaviour, without neglecting the upcoming risks associated with increasing system complexity.

3.1 Explanation

There are several excellent definitions available for CPS. The following quotes present well-suited definitions for this thesis:

> In a cyber-physical system (CPS), some elements are computational and some involve interactions with the physical environment, integrating "computation, communication, sensing, and actuation with physical systems to fulfil time-sensitive functions with varying degrees of interaction with the environment, including human interaction". A CPS incorporates multiple connected systems, producing a system capable of developing an awareness of its physical environment and context, making decisions based on that information, and enacting work that can effect changes in its physical environment.
> - *Romanosky & Ishikawa, Trustworthy Cyber-Physical Systems Engineering [5]*

> A cyber-physical system consists of a collection of computing devices communicating with one another and interacting with the physical world via sensors and actuators in a feedback loop.
> - *Rajeev Alur, Principles of Cyber-Physical Systems [6]*

As depicted in figure 3.1, a system that consists of both cyber and physical components is referred to as a cyber-physical system. It represents a generalisation of an embedded system [6]. The cyber part of a CPS comprises all digital components such as software applications, computing devices, communication networks, and algorithms. The physical part of

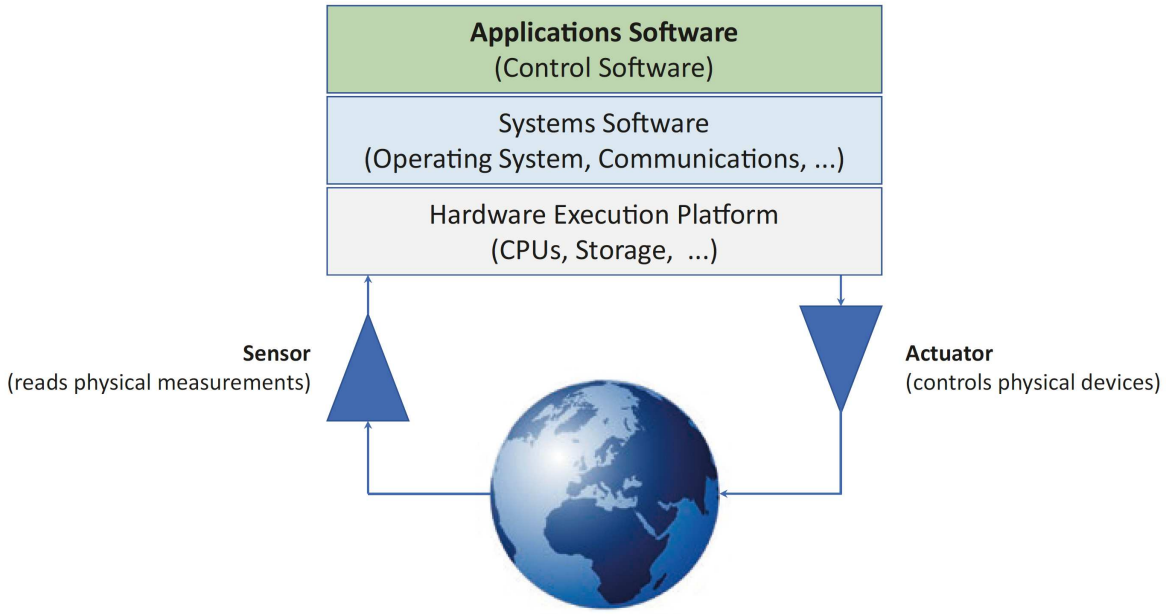


Figure 3.1: Cyber-Physical System [7]

a CPS encompasses all components that directly interact (actuators) or measure (sensors) the physical world [7]. Sensor readings are evaluated by the cyber part, which then triggers specific actions for the actuators.
Cyber-Physical Systems hold the promise of significant economic and societal benefits in domains such as aerospace, production automation, transportation, and healthcare [6]. "Future cyber-physical systems will contribute to safety, efficiency, comfort and human health like never before. They will therefore contribute to solving key challenges of the modern society, such as the ageing population, limited resources, mobility, or the shift towards renewable energies, to name but a few fundamental fields of application." [8]

However, due to the dependence on CPS and their direct interaction with the physical world, *they pose direct risks to human life, property, and the environment*. Additionally, outages can bring critical supply chains to a halt.
Therefore, it is crucial to thoroughly examine the safety and security-critical aspects of CPS.

3.2 Safety- and Security-Critical Aspects

As previously defined, cyber-physical systems directly interact with the physical world, creating an "intersection ... of the physical and the cyber" - Edward A. Lee. In this context, the control software/application can measure and modify environments within the physical world.

Consequently, a malfunctioning CPS can pose a direct threat to the physical world and the entities residing within it. Conversely, a well-functioning CPS can offer significant benefits to its users.

Consider the advancements in agricultural machinery that have been summarised and evaluated for their benefits and challenges in a scientific paper by Jilin University and Politecnico di Milano [9]. Since the invention of the first industrial robots in the 1950s, CPS have expanded into the agriculture domain, experiencing rapid evolution [9]. The paper states that in order to address the challenges posed by the aggravation of population ageing and the limitations of traditional labour-intensive and risky farm work, there is a need

for empowering agricultural activities through more automated control [9]. For instance, CPS in agriculture, such as automated harvesters, enable a single farmer to supply food to a significantly larger population compared to pre-modern farming practices. When CPS function correctly, they offer immense benefits. However, in the event of malfunction, they could pose potential risks to human workers, lead to crop damage, or even cease functioning altogether.

The food industry relies heavily on CPS, and widespread unavailability of these systems could result in severe food shortages and potentially lead to famines.

Hence, it is crucial to implement rigorous and robust methods to ensure the trustworthiness of these systems [6], ensuring that the risks during malfunction do not outweigh the benefits during normal operation. This thesis specifically focuses on the safety and security attributes of each CPS as indicators of trustworthiness.

For a system to be considered trustworthy by end-users, it must not only match the safety and security levels of the predecessor it replaces but exceed them significantly. For instance, in the case of autopilot systems replacing traditional driving (level 0-2 driving automation), the autopilot system must exhibit near-perfect levels of security and safety to convey trust in end-users, even though traditional driving experiences more accidents due to human error. Tesla, an automobile manufacturer, published their annual vehicle safety report, as depicted in Figure 3.2. The report demonstrates that vehicles utilising Tesla's autopilot are considerably less likely to be involved in accidents compared to their own vehicles with the autopilot turned off or the average vehicle in the United States. In 2022, a Tesla vehicle using autopilot was involved in an accident on average every 5.695 million miles driven, while the United States average for vehicle accidents was one accident every 0.652 million miles driven [10].

Nevertheless, the latest AAA survey on drivers' attitudes towards self-driving vehicles indicates a decreasing level of trust in autonomous vehicles [11]. Numerous other surveys also demonstrate significant scepticism among the majority regarding their trust in autonomous vehicles, as summarised in a report by the Advocates for Highway & Auto Safety [12].

Therefore, the CPS manufacturers have to work more diligently on increasing trustworthiness for their systems as Nazila Gol Mohammadi said: "Trustworthiness (...) is a key success factor for (CPS) use and acceptance by end-users."

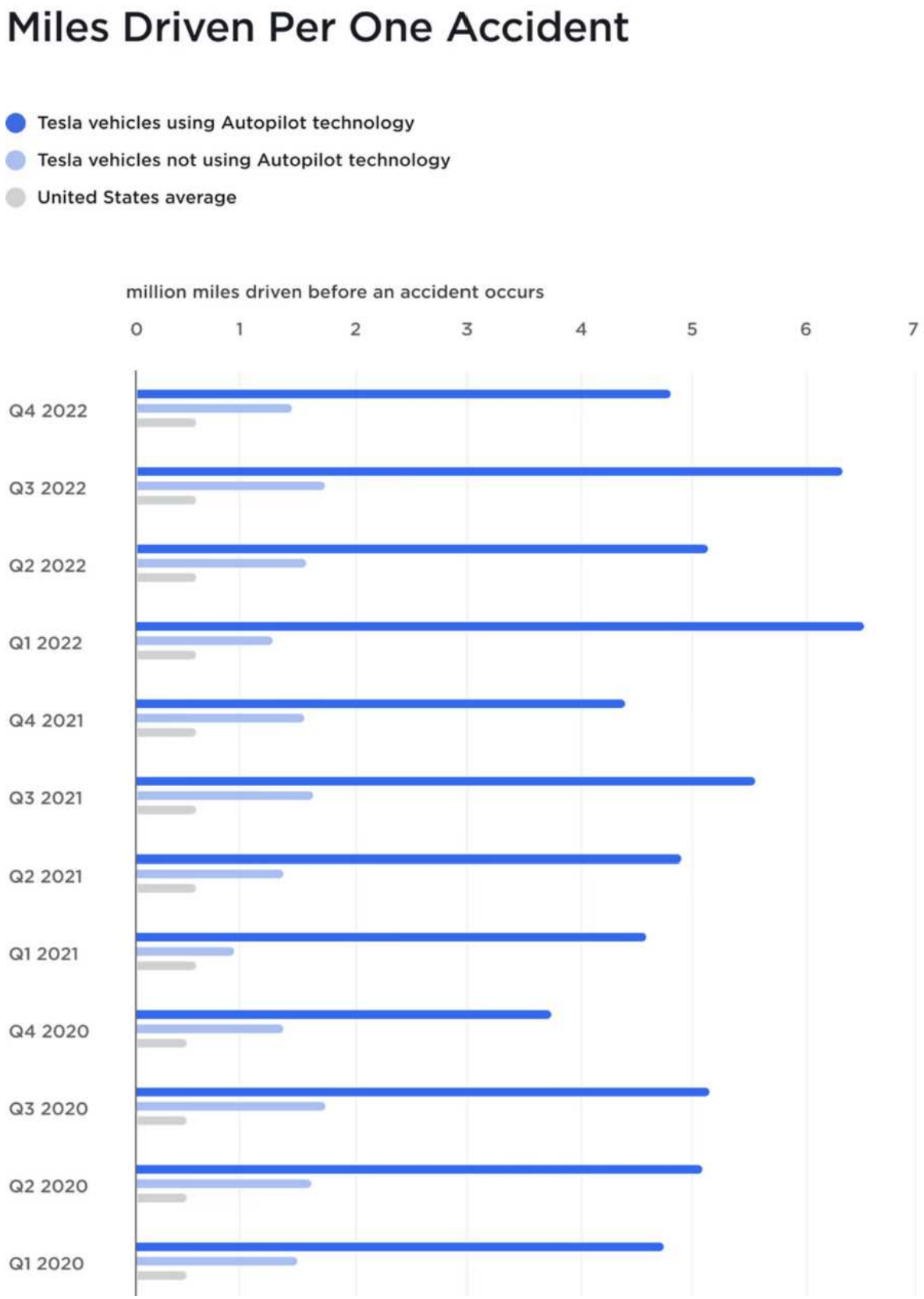


Figure 3.2: Tesla Vehicle Safety Report [10]

3.3 Risk

This thesis frequently uses the word risk. Therefore, it makes sense to define this often colloquially used term more academically in regard to security incidents and safety accidents. *Risk can be defined as the evaluated threat to a system based on the frequency of occurrence (probability of happening) and the impact or magnitude of the potential damage (degree of harm)*. It refers to the assessment of the potential harm or danger exposed by this system, taking into account both the likelihood of the event happening and the severity of its consequences.
Contrary to common belief, the incident or accident risk is not solely determined by the probability of an event happening. Similarly, the mere potential for an incident or accident to result in fatalities does not necessarily imply a high risk. Instead, risk assessment involves a comprehensive evaluation of both the likelihood of an event and the extent of its potential damage. Therefore, an accident that is very unlikely to occur and has a medium damage potential would exhibit a rather low risk.
There are two options how risk can be assessed; *quantitative* and *qualitative*.

3.3.1 Quantitative Risk Assessment

One approach to assessing risk is through quantitative risk assessment, which involves attempting to quantify risk using a calculation formula. A common formula used is as follows:

$$R = p_O \cdot D_P$$

R - risk
p_O - probability of occurrence
D_P - damage potential

Authorities and regulators strongly favour the quantitative approach for certifying critical CPS due to its inherent objectivity and enhanced transparency in assessing risks. However, there are several challenges associated with the quantification approach to risk assessment. Determining probabilities for certain events, such as the occurrence of a bit flip caused by a single event effect, can be challenging due to their low likelihood. Moreover, probabilities may be influenced by uncontrollable factors like the motivation of potential attackers. Additionally, relying on statistical data or reliable experience to support the quantification process can introduce inaccuracies. When these inaccuracies are multiplied, they can have a significant impact on the resulting risk assessments, potentially leading to erroneous conclusions. "Quantitative approaches are very rarely used, as they require the existence of a formal model describing the system under study. Attempting to analyse (...) risk has been shown quantitatively to be either infeasible or inadvisable in most real-world situations." [13]

3.3.2 Qualitative Risk Assessment

Given the limitations of quantitative risk assessment, the industry commonly adopts a qualitative risk assessment approach. This method employs distinct classes for the probability of occurrence and the damage potential, as shown in the table below 3.1. Properly defining these classes is crucial to achieve consistent classification. Each threat or failure is assigned a probability class and a damage class, which then determines its risk class within a risk matrix.
The example risk matrix 3.2 visually represents the classification of risks based on their probability and damage potential. The risk classes can be categorised as acceptable or unacceptable, aiding in the decision-making process for risk mitigation. In the provided risk matrix 3.2, the low and medium risk classes are considered acceptable, while the high and extremely

p_O	D_P
very low	insignificant
low	minor
moderately	medium
high	large
very high	catastrophic

Table 3.1: Qualitative Classes for Probability and Damage

high risk classes are deemed unacceptable. This distinction is illustrated by the black "*acceptance line*" drawn diagonally across the matrix, which serves as a predefined threshold for accepting or rejecting risks during the risk management process.

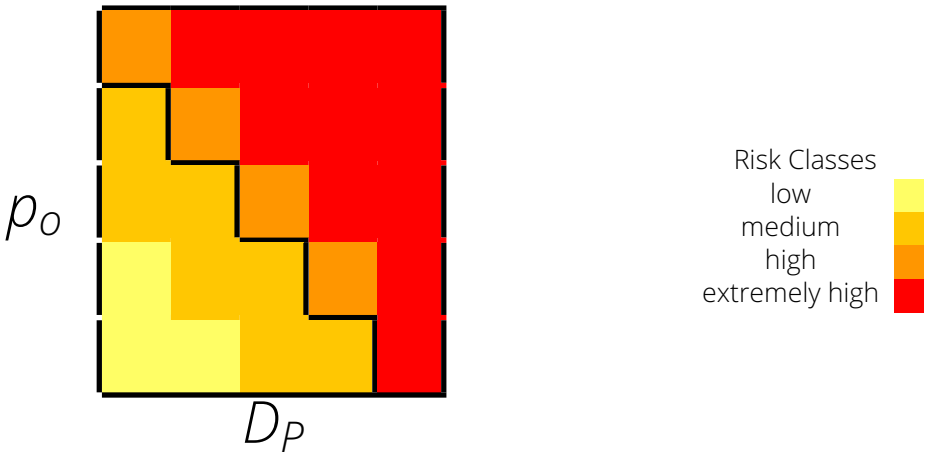


Table 3.2: Risk Matrix

3.3.3 Risk Reduction Mechanisms

To achieve an acceptable level of risk, engineers have two primary options: *reducing the damage potential or decreasing the probability of occurrence*.
Reducing the probability of occurrence involves eliminating vulnerabilities within the system. In an ideal scenario, a system without any vulnerabilities susceptible to exploitation or prone to failure would pose no risk on its own. By minimising or eliminating potential attack vectors or failure mode, the probability of occurrence can be effectively reduced for each individual risk.
On the other hand, the damage potential can be reduced by implementing additional safety and security measures. For instance, the inclusion of a safety switch on a lawnmower that requires continuous pressure on the control bar for operation significantly reduces the damage potential. This safety mechanism ensures that the lawnmower cannot operate without proper supervision, thereby minimising the risk. Furthermore, if certain functionalities within the system possess an unmanageable damage potential, engineers must decide not to implement those functionalities, ultimately reducing the overall damage potential.
By employing these risk reduction mechanisms, engineers strive to bring individual risks to an acceptable level, enhancing the security and safety of cyber-physical systems.

3.3.4 Acceptable Residual Risk

Risk is an inherent and inevitable aspect of every system, characterised by its imperfections and the absence of absolute safety or security. Consequently, when implementing an engineered system, it becomes necessary to set a threshold that defines the level of risk that engineers are willing to take, known as the acceptable residual risk. Without such a threshold, the system would remain inoperable. The National Institute for Standards and Technology (NIST) has numerous definitions within their glossary to explicate the term residual risk [14]:

- "Portion of risk remaining after security measures have been applied."
- "The potential for the occurrence of an adverse event after adjusting for the impact of all in-place safeguards."
- "Risk that remains after risk responses have been documented and performed."

3.4 Engineering Principles

The design and development of trustworthy CPS require adherence to engineering principles that contribute to safety and security. These principles, also known as industry best practices, distil *proven, reliable knowledge* from systems engineering and successful architecting into *precise, actionable, and enforceable guidance* for the development process [7]. They serve as a valuable tool for CPS engineers in creating systems with an acceptable residual risk, as presented in the previous section.
Safety- and security principles are an essential subset of engineering principles. They guide building a safety- and security-aware CPS architecture, the implementation and operation by providing blueprints and patterns that have been proven to be suitable for similar systems.
Therefore, the architecture can be defined as: "The fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" [5], where principles play a central role during the design-phase and the evaluation of the system.
They often offer *generic solutions to recurring architectural or design problems* encountered in CPS development projects [7].
Importantly, engineering principles are formulated to be neutral, making them usable in various project contexts. If these principles were context-specific, they would become rigid and challenging to implement across different projects. Therefore, engineers must apply a context-based interpretation when utilising the selected principles in their specific CPS development context [7]. One significant advantage of relying on established, rigorously evaluated engineering principles, besides saving time and work effort, is that they have been evaluated and refined by experts over time. This extensive scrutiny makes it more unlikely that these principles contain obvious flaws compared to solutions solely developed for individual projects. By incorporating these widely accepted principles, CPS engineers can benefit from the collective expertise and experience of the engineering community.
In line with the goal of contributing the engineering principles toolkit, this thesis suggests and defines a new principle: the generic protective shell. Following the design science methodology, the artefact created in this thesis aims to safeguard the CPS during run-time by providing a comprehensive and adaptable framework, the generic protective shell, for ensuring safe and secure operation of CPS. This will be elaborated further in chapter 4 Protective Shell.

By adhering to safety and security principles, it enables CPS engineers to create robust and trustworthy CPS with effective risk management and protection against potential threats

and failures. These principles ultimately contributing to the advancement and maturity of the CPS field.
Following, an introduction for safety principles and security principles is given, with one example respectively. It is important to note that this introduction does not claim to be exhaustive, as there exists a vast amount of information available on this subject. For a more detailed examination and an extensive list of these principles, it is recommended to refer to Professor Furrer's book, "Safety and Security of Cyber-Physical Systems" [7].

3.4.1 Safety Principles

Safety principles refer to the fundamental guidelines and practices aimed at ensuring the safety and integrity of systems throughout their design, deployment, and operation phases. The goal is to prevent accidents, minimise risks, and protect individuals, assets, and the environment from harm.

Definition Safety:

> Safety is the state of being protected against faults, errors, failures, or any other event that could be considered non-desirable to achieve an acceptable level of risk concerning the loss of property, damage to life, health or society, or harm to the environment.
> - *Frank J. Furrer, Safety and Security of Cyber-Physical Systems [7]*

A famous example is the concept of "no single-point-of-failure". This principle emphasises the use of a robust structure that facilitates defence-in-depth [5]. In CPS, where failures can have significant consequences, this principle promotes the design and implementation of systems that *do not rely on any single component* prone to fail for safety. By adopting a defence-in-depth approach, multiple layers of protection and redundancy are employed to ensure that a failure in one component or layer does not compromise the overall safety of the system. This principle is deeply ingrained in the safety engineering practices of CPS and is crucial for mitigating risks and enhancing system resilience. It contributes to the system's fault tolerance and it's fail-safe design.
Fault tolerance involves designing systems that can continue to operate and provide essential functionalities even in the presence of component failures or environmental disturbances. Fail-safe design focuses on ensuring that system failures lead to safe and controlled states rather than hazardous, escalating situations. Therefore, a hazard analysis needs to be completed to identify and assess potential hazards within the system and developing appropriate mitigation strategies.

The utilisation of safety principles in CPS is often guided by regulatory frameworks like the International Organization for Standardization (ISO) 26262 for road vehicles aiming to achieve functional safety. These standards provide comprehensive guidelines for the identification, analysis, and management of safety-related risks throughout the life cycle of CPS, ensuring a systematic and rigorous approach to safety engineering.

3.4.2 Security Principles

Security principles refer to the fundamental guidelines and practices aimed at protecting systems from unauthorised access, malicious attacks, and potential data breaches. The primary objective of security principles is to establish a robust and resilient security posture that safeguards the *confidentiality, integrity, and availability* of CPS components, data, and

operations.

Definition Functional- & Information Security:

> Functional security protects the software-system from malicious, infiltrated code, both from the outside and from the inside of the organisation.
> Information security protects the confidentiality, integrity, and availability (CIA) of computer system data and information from unauthorised and malicious accesses.
> *- Frank J. Furrer, Safety and Security of Cyber-Physical Systems [7]*

One security principle example widely adopted is the concept of a "zero-trust architecture". This strategic approach to cybersecurity that promotes the *continuous validation of every digital interaction*. It eliminates implicit trust by leveraging strong authentication methods, network segmentation and granular, "least access" policies [15]. This principle acknowledges that in CPS, no entity should be intrinsically trusted solely because it is being part of the CPS architecture, or its association with the system.

Security principles are often governed by various standards and regulatory frameworks, such as the National Institute of Standards and Technology (NIST) Cybersecurity Framework and the ISO 27001. These standards provide guidelines and best practices for, besides others, establishing robust security controls and completing effective risk management processes. Compliance with these principles and standards is crucial to mitigating security risks, safeguarding critical infrastructure, and protecting against emerging cyber threats in the rapidly evolving CPS landscape.

3.5 Cyber-Physical System of Systems (CPSoS)

In today's technologically advanced society, the demand for increased functionality and the great expectations towards modern systems require the engineers to compose the final system through the integration of multiple constituent systems. This interconnected arrangement, known as a *system-of-systems* (SoS), is formed by the collaboration of individual systems to deliver the high functional specifications.

Definition System-of-Systems:

> A system-of-systems (SoS) is a set or arrangement of constituent systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities.
> *- Frank J. Furrer, Safety and Security of Cyber-Physical Systems [7]*

The unique capabilities are the ones that no single constituent system would achieve on its own, but through cooperation towards a higher objective, they can be achieved nonetheless. A *cyber-physical system of systems* (CPSoS) is a specific type of SoS that incorporates at least one cyber-physical system (CPS) interacting with the physical world within its system arrangement, as shown in figure 3.3. Likewise to SoS, CPSoS enables the achievement of higher objectives through collaboration among various constituent systems. For instance, in the context of driver safety in modern vehicles, no single system can provide the necessary level of safety independently. However, through the cooperation of multiple systems such as the supplemental restraint system (SRS) airbags, anti-lock braking system (ABS), and autonomous emergency braking (AEB), the higher objective of driver safety emerges.

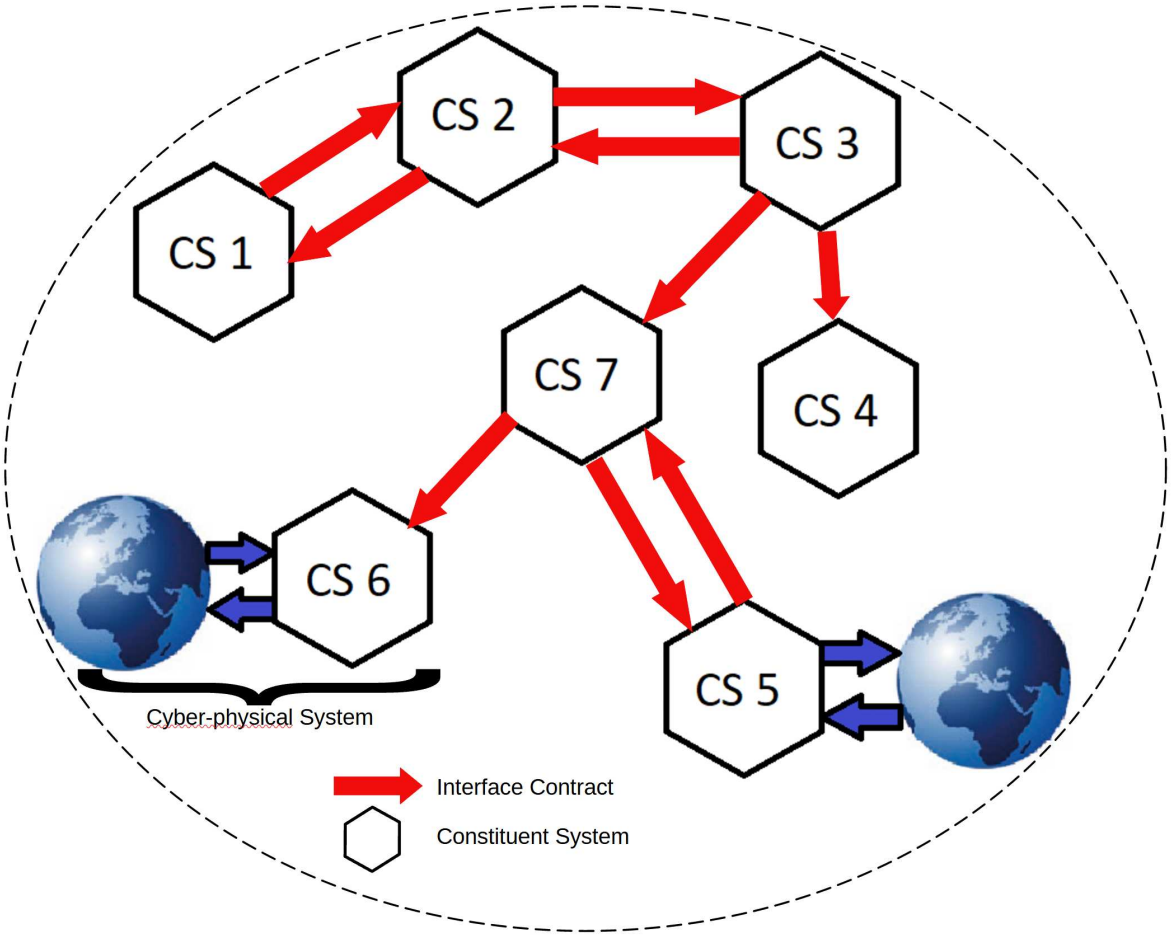


Figure 3.3: Cyber-physical System of Systems

3.5.1 Emergence

The main motivation for arranging systems in a cooperative mesh is the emergence of new properties or behaviours.

Definition Emergence:

> An emergent property/behaviour is a property, behaviour, or aggregated information, which a collaboration of constituent systems have, but which the individual constituent systems do not have.
> *- Frank J. Furrer, Safety and Security of Cyber-Physical Systems [7]*

The integration of multiple systems with useful functionalities enhances flexibility but also introduces greater complexity to the overall system. Consequently, *emergent behaviours may lead to unintended outcomes that individual systems alone would not cause.* Even if the individual systems are thoroughly understood, with each failure mode adequately handled and mitigated, the collaboration of systems can introduce new failure modes.

> Trustworthiness is a holistic property that calls for the co-engineering of safety and cybersecurity, among other qualities. It is not sufficient to address one of these attributes in isolation, nor is it sufficient simply to assemble constituent systems that are themselves trustworthy. Composing trustworthy constituent systems may imply an untrustworthy SoS.
> *- Flavio Oquendo, 2015*

The emergence matrix 3.3 illustrates the four distinct quadrants that arise in every SoS and must be considered by system developers.

	Desirable (positive)	Undesirable (negative)
Expected Emergence	Q1 = Reason for building the CPSoS	Q2 = Risk-analysis, understand and mitigate
Unexpected Emergence	Q3 ?	Q4 = Source for Safety accidents, Security incidents

Table 3.3: Emergence Matrix [7]

Quadrant 1 (Q1) represents the desired and expected outcomes of system collaboration, which depicts the main reasons for arranging the SoS. In the previously mentioned example, the enhancement of driver safety through multiple systems working together exemplifies the positive impact achieved. All systems contribute, to a varying extent, to the driver safety in the event of an accident trying to safeguard him from harm.

Quadrant 2 (Q2) encompasses the expected but undesired outcomes resulting from system collaboration. These must be identified during the development phase through rigorous risk analysis. *Engineers hold the responsibility of mitigating such negative emergent behaviours*, ultimately preventing their occurrence. In the context of driver safety, an example of an expected but undesired outcome involves the emergency braking system (AEB) triggering an abrupt stop, catapulting the driver forward. If the driver's head is in close proximity to the airbag casing when an impact occurs immediately after emergency braking, the airbag may explosively inflate, causing severe injury. To mitigate this anticipated undesired outcome, engineers implemented a seatbelt system to restrain the driver's body, allowing safe inflation of the airbag.

Quadrant 3 (Q3) comprises outcomes that were not expected by the systems engineers but still add value to the overall system. Although rare and unexpected, these outcomes are desirable and thus require no mitigation measures. Nonetheless, it should be noted that they are of great interest for the engineers. These outcomes were overlooked during development and need to be analysed to extract valuable insights in further understanding the inner workings of the system.

Quadrant 4 (Q4) represents the emergence of unexpected and negative outcomes resulting from the collaboration of constituent systems. They serve as the primary source of safety accidents and security incidents in CPSoS scenarios, even when CPS engineers diligently follow safety- and security-aware development processes. *Such harmful emergence cannot be effectively addressed through conventional risk management processes due to its unexpected nature*. Typically, these unexpected emergent behaviours have a very low probability of occurrence. If, however, they do occur during testing and can be replicated, their causes can be identified, and potential mitigations explored.
Chapter 5 analyses several accidents and incidents that originate from Quadrant 4 outcomes. A prominent example is the Toyota "Unintended Acceleration" 5.4 safety accident. The accident was triggered by an unforeseen situation in which the Toyota Electronic Throttle Control System (ETCS) experienced memory corruption in its Static Random-Access Mem-

ory (SRAM) due to either a single event effect or software defects, such as stack buffer overflows. Regrettably, the Toyota engineers had not anticipated this failure mode and consequently did not implement essential mitigation mechanisms, such as error detection and correction (EDAC) for SRAM. As a result, unexpected negative behaviours occurred during run-time, leading to the suspension of critical ETCS tasks and eventually causing an electronically stuck acceleration pedal, which resulted in numerous accidents.

Unfortunately, most instances of unexpected emergence manifest during normal system operation (i.e. after the system got deployed to its operation environment), leading to safety accidents and security incidents in an uncontrolled domain. Several factors contribute to this, such as the increased likelihood of rare events occurring with higher numbers of the same CPS being operated or the diverse ways in which different operators handle the CPS, which cannot be adequately simulated during testing.
When unexpected emergent behaviours occur, the system may mishandle the unknown situation, triggering a cascade of negative effects that leads to CPS failure and subsequent accidents or incidents. Although quick manual intervention by the CPS operator can potentially halt such cascading effects, human reaction times are relatively slow compared to the rapid feedback-loop iterations of the CPS. Consequently, human intervention often fails to prevent or even detect the impending accident or incident.
This emphasises the need for a tool supporting the CPS operator to react to those emergent failures or attacks more efficiently and/or initiating corrective and protective actions on it's own. This tool is called a protective shell and will be extensively elaborated in the subsequent chapter.

4 Protective Shell

This section introduces and elaborates the concept of a protective shell on the current state-of-the-art. After the introduction, the system architecture is presented, which can be divided into the design-time architecture and the run-time architecture. The importance of sound architecture for building trustworthy systems is also emphasised.
Furthermore, the concept of run-time monitoring is explained, followed by a detailed elucidation of the protective shell, its definition and capabilities. Lastly, the expectations and goals towards the protective shell will be stated.

4.1 Explanation

Modern CPS have become increasingly complex, and this trend is expected to further accelerate in the future. The main reasons for increasing complexity is the rapid growth of functional requirements for CPS and their *rising interconnectedness in obscure networks*. Manufacturers are under pressure to integrate CPS into CPSoS to fulfil these high functional requirements, thereby rising the end systems complexity. The complexity of a system directly correlates with its attack surface, the number of failure modes it exhibits, and the probability of unpredictable and unexpected emergent behaviour. These factors contribute to the growing safety and security concerns in modern CPS.
In the past, the system quality properties - such as safety and security - could be ensured to a satisfactory extent during the development process of simpler CPS with lower complexity. However, in today's complex systems, ensuring all quality properties in all operating conditions during design time is practically impossible [2]. This does not imply that the development process for the design architecture needs to be replaced; rather, it highlights the *limitations of developing trustworthy systems solely during the design phase*. There is a need to improve the overall system architecture during operation to ensure the required quality properties.
The history of system engineering has shown that system quality properties strongly depend on the underlying system architecture [2]. To build trustworthy and modern CPS, it is necessary to improve the system's architecture beyond the implementation of industry-standard engineering principles and completing risk management processes.
In 2023, Hermann Kopetz presented five results that are unattainable during system design. These outcomes, referred to as the "*impossible results*", contain various challenges that cannot be fully resolved during the development process. The impossibilities are as follows:

It is impossible ...

1. to find all **design faults** in a large system
2. to find all **implementation faults** in a large system
3. to foresee all **operational conditions** of the system in its deployed environment
4. to mitigate all consequences of **execution platform failures**, including hardware failures, network outages, partner-system unavailability
5. to fully identify **emergent properties** or **emergent behaviour** while assembling the system

This chapter explores the concept of a "protective shell" as an additional component in the run-time architecture of CPSs to enhance their overall safety and security and, at the same time, to address the aforementioned impossible results during the run-time phase.

4.2 System Architecture

The overall system architecture can be divided into the *design-time architecture* and the *run-time architecture*.
The *design-time architecture* is created during the development process, where engineers aim to eliminate known vulnerabilities, follow engineering principles, and complete risk management processes. However, the high complexity of modern CPS leads to inevitable vulnerabilities within the design-time architecture, arising from emergent behaviour, unknown weaknesses, or disregarded best practices. These vulnerabilities can be intentionally exploited by malicious threats, leading to security incidents, or they can result in safety accidents due to unexpected operating conditions. Unfortunately, the refined and sophisticated development process leaves little room for safety and security improvements when followed diligently by the engineers.
On the other hand, the *run-time architecture* of CPS can be further improved to address vulnerabilities present in the deployed system. As shown in figure 4.1, once the development process is completed, and the system is deployed in its operational environment, it becomes a run-time system that may still contain vulnerabilities. The introduction of additional components, such as active run-time monitoring, can enhance the run-time architecture and provide added safety and security to the system.

4.3 Run-Time Monitoring

Run-time monitoring involves constant evaluation of the current system behaviour in comparison to the expected and desired behaviour. As depicted in figure 4.2, the desired behaviour is defined through a combination of rules, policies, models, and functional specifications. Defining such static rule-sets requires consultations with domain experts and the responsible CPS engineers, necessitating additional work effort. However, this process can also be automated by training a behaviour baseline through information retrieval from operational data and log files, using appropriate machine learning algorithms and a vast amount of such operational information.

The run-time monitor embraces the run-time system and attempts to protect it from the impact and the consequences of threats and failures, preferably in real-time [2].

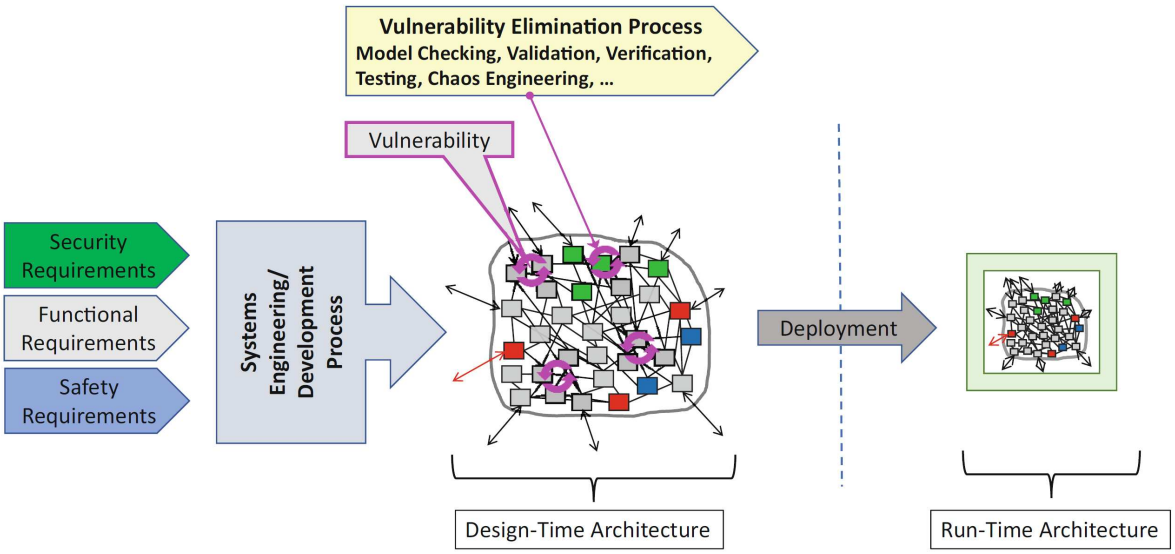


Figure 4.1: Design-Time Architecture Deployment [2]

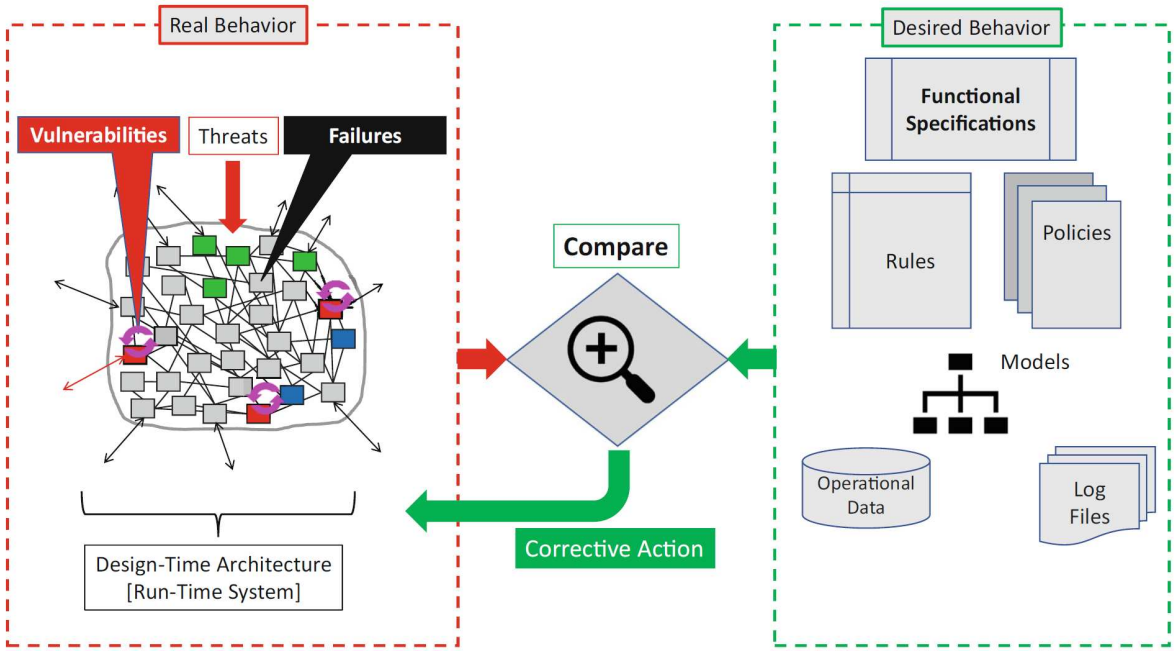


Figure 4.2: Run-Time Monitoring [2]

To identify deviations from the expected behaviour, run-time monitoring utilises basic validation and comparison as well as more advanced anomaly detection based on artificial intelligence algorithms (which will get elaborated in chapter 6). Detection mechanisms are a crucial element within the protective shell and can be further configured to suit the CPS requirements exceptionally.
The concept of implementing an additional component similar to the protective shell is not entirely new. Lance Eliot has presented the idea of having an "AI Guardian Angle Bot" supervise underlying deep AI systems to improve their trustworthiness in his book [1]. This situation is similar since deep AI tend to utilise algorithms and structures that lack intrinsic explainability, which can lead to unexpected behaviour. Furthermore, there are already modern CPS that use deep AI, spanning even stronger parallels to Eliot's book.

4.4 Definition

A protective shell is an

> Independent hardware/software layer overlaid on a system using additional information, such as supplementary sensor data or log files, to detect, analyse, interpret, and mitigate potentially dangerous or unsafe system behaviour, either in realtime, near-realtime, or batch processing.
> *- Frank J. Furrer, Master Thesis Topic*

Figure 4.3 illustrates the basic architecture of a protective shell. *The cyber-physical run-time system forms the core of the run-time architecture*, completely embraced by the protective shell, which separates it from interfaces to the physical world and external digital networks. The protective shell has *access to all data streams passing through it*, acting as a membrane. This includes, but is not limited to, sensor data, firmware updates, network access, application signals to actuators, operational data, and log files. Additionally, it may incorporate supplementary sensors, precalculated behaviour models, functional specifications, rules, and policies obtained from domain experts.

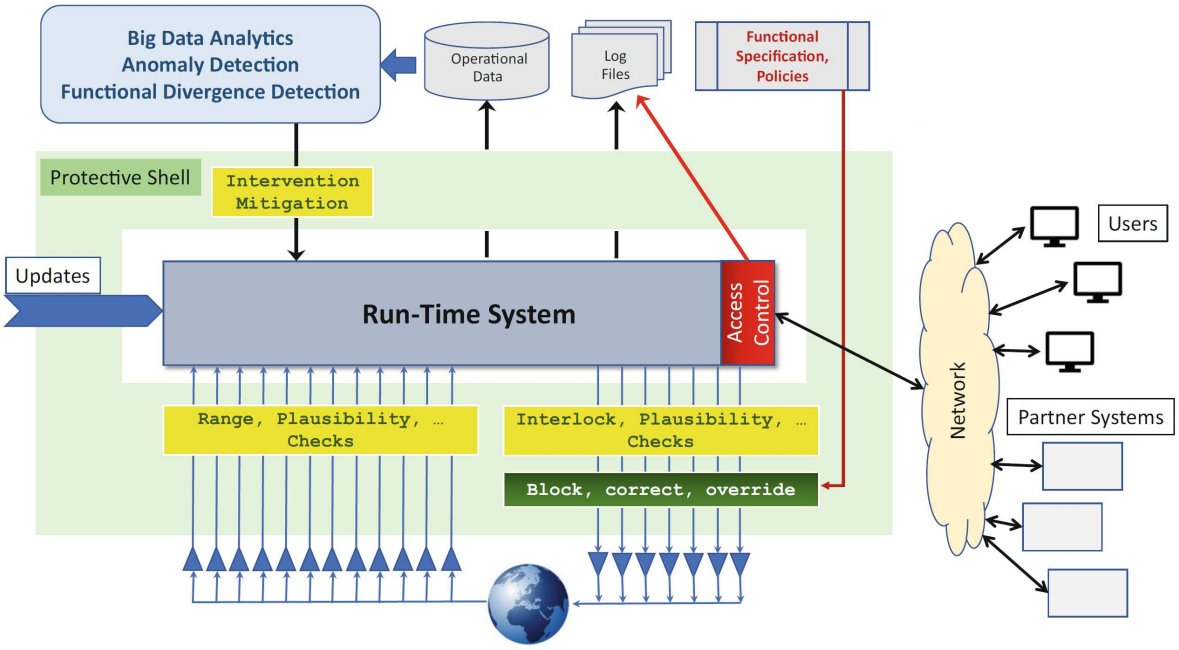


Figure 4.3: Protective Shell [2]

With access to such extensive information, the protective shell can utilise traditional techniques such as range checks for actuators and sensors, as well as plausibility validation. However, its true strength lies in employing anomaly detection through machine learning and artificial intelligence. This allows the protective shell to identify if the underlying CPS is heading into a failure mode or if it remains in a safe operational state. The protective shell may also detect if the underlying CPS slowly deviates from the predefined desired behaviour over time.
When an anomaly or a mismatch between desired and actual behaviour is detected, and the CPS fails to initiate remediation actions on its own, the protective shell assesses the situation, jumps in as a last line of defence and executes corrective actions by manipulating the data streams passing through it or directly accesses the CPS.

4.5 Expectations / Goals

The protective shell concept has two primary objectives: firstly, to *mitigate vulnerabilities that may have been overlooked during the CPS development process*, and secondly, to *address undesired emergent properties* during system operation. As illustrated in figure 4.4, the total amount of vulnerabilities within a system decreases over the course of the development process due to rigorous safety and security measures. This process involves thorough risk analysis, identification, assessment, and mitigation steps, including the validation and verification of the architectural design and sophisticated testing methodologies.
However, vulnerabilities may persist from various sources, specifically originating from the second quadrant of the emergence matrix discussed in section 3.5.1. These vulnerabilities correspond to expected but undesired behaviour. Engineers might consciously decide to tolerate these vulnerabilities, assuming that the overall risk (probability of occurrence in respect to the damage potential) exposed by these vulnerabilities is negligible. Alternatively, ineffective mitigation strategies might be employed for known vulnerabilities, or standardised risk management processes might be disregarded. Collectively, these factors contribute to the presence of vulnerabilities that should ideally have been eliminated during a sophisticated development process, yet manage to infiltrate and manifest within the CPS's design architecture.
Nevertheless, vulnerabilities may also originate from the fourth quadrant of the emergence matrix, representing unexpected negative behaviour. These vulnerabilities cannot be eliminated during the development process and are not directly a consequence of disregarded risk management processes. Since they are unexpected and unknown, no specific mitigation technique is available other than broadly applying industry best practices to all system components.
As depicted in figure 4.4, such vulnerabilities persist even after finishing the developmental process. Particularly in complex CPS, this lingering number of vulnerabilities exposes an unacceptable level of residual operational risk. This reality necessitates an additional approach, concept or tool to further mitigate vulnerabilities present within the design-time architecture. *The protective shell, with its run-time monitoring and real-time intervention capabilities, emerges as a promising mechanism to counteract unexpected dangerous behaviour* [2] at the very instant it surfaces. Currently, mitigation is often discovered after the threat or failure has already surfaced, providing engineers the option to analyse the operational data. In contrast to this retrospective view, the protective shell could provide immediate protection against vulnerabilities exploited by threats and failures that were previously unknown.
By implementing a protective shell, it is expected that CPS can better address vulnerabilities arising from emergent behaviour and reduce their negative impact as well as consequences during run-time, ultimately enhancing the safety and security of these systems.
This marks a significant advancement in the field of CPS architecture, since the currently applied, rather static tools within the development process of the design-time architecture do not provide sufficient protection capabilities against quadrant four negative behaviour for complex systems. As indicated in figure 4.4, a substantial reduction in total vulnerabilities is achievable through a diligent adherence to safety- and security-aware development procedures, complemented by a proficiently configured and implemented protective shell optimised for the embraced CPS.
However, it is important to acknowledge that achieving absolute or perfect safety and security remains an unattainable objective. It is essential to recognise that the resulting system, despite these effective measures, may still contain "deeply hidden" vulnerabilities. Nonetheless, with this reduced amount of vulnerabilities, the system now possesses a higher probability of falling within the threshold of acceptable residual risk.

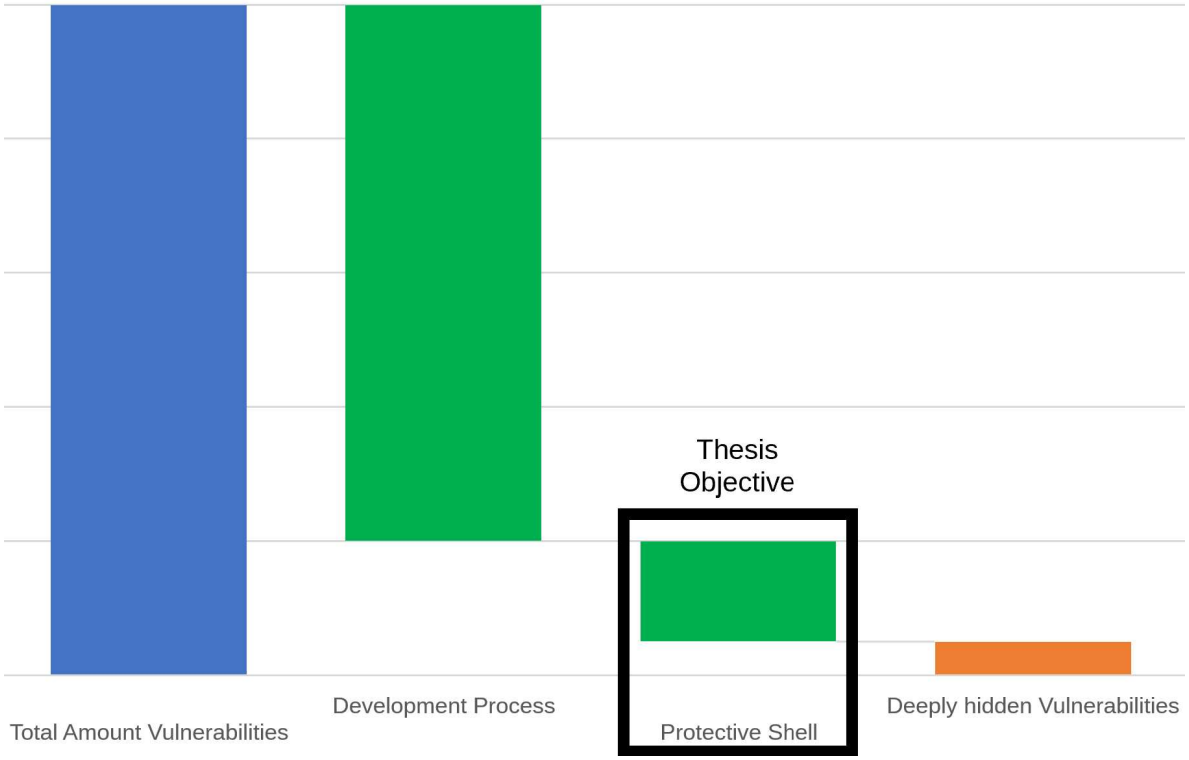


Figure 4.4: Vulnerability Mitigation and Elimination

5 Specific Protective Shells

This chapter applies the previously discussed theoretical aspects of a protective shell to practical scenarios with case studies. To illustrate the implementation of protective shells, a selection of well-documented safety accidents and security incidents involving CPS is examined as listed in table 5.1, that could benefit from protective shell implementations. These accidents and incidents are adequately documented from reliable sources to enable an analysis of the underlying causes, identification of vulnerabilities, and drawing of conclusions with a high degree of certainty. Furthermore, they were chosen from a wide variety of sectors to have a good overall representation without large biases.

Name	Incident / Accident	Year	Sector	Outcome	Main Vulnerability
Boeing 737 Max MCAS	Accident	2018/19	Aviation	two crashes, 346 fatalities	Single Point of Failure
Therac-25	Accident	1985-87	Health	at least 6 fatalities, several injuries	Software Race Condition
Stuxnet	Incident	2010	Research	~1000 uranium enriching centrifuges destroyed	Neglected Range Validation
Toyota "Unintended Acceleration" ETCS	Accident	2009-11	Automotive	at least 89 fatalities, 57 injuries	Single Point of Failure
Jeep Cherokee Hack	Incident	2015	Automotive	1.4 million vehicle recall by manufacturer	Zero-Trust Architecture Violation
Ukrainian Power Grid Attack	Incident	2015/16	Cr. Infrastructure	widespread power outages affecting >255k customers	Weak Network Segmentation and IAM
Airbus A400M FADEC	Accident	2015	Aviation	one crash, 4 fatalities and 2 injuries	No Software Integrity Checks

Table 5.1: Overview chosen Safety Accidents and Security Incidents

After identifying the key weaknesses in each CPS, a specific protective shell per CPS will be developed to introduce an additional protection layer in the defence-in-depth strategy, aiming to reduce the risks to an acceptable level. However, it should be noted that the specific protective shells will not be refined to an implementable state out-of-the-box; rather, the protective mechanisms, characteristics and capabilities will be thoroughly explained.
All incidents and accidents discussed in this chapter were based on CPSs with unidentified or miscalculated risks during development. The analysis now benefits from the availability of post-event data, such as operation data and log files, allowing for more accurate identification of these risks. In a typical CPS development lifecycle, engineers would implement mitigation measures directly to the CPS design-time architecture to eliminate the newly identified vulnerabilities. However, in this *theoretical academic scenario, the underlying CPS design-time architecture remains unchanged*, and the mitigation techniques are solely implemented in the protective shell. This approach enables the evaluation of the protective shell's effective-

ness and highlights which mitigation capabilities are frequently used, uncovering similarities among different protective shells.
Mitigation techniques can leverage the entire capabilities shown in the protective shell architecture in figure 4.3. For this theoretical scenario, it can be assumed that domain experts defined the desired behaviour with functional specifications, rule-sets and policies completely, consistently and without any (logic) flaws. Additionally, it can be assumed that the machine learning algorithm retrieved the ideal model from operation data and log files for exceptional anomaly detection. Risks that may arise due to vulnerabilities outside the CPS design-time architecture's influence are mentioned during the weakness analysis but are considered out-of-scope for the development of the protective shell, since it has no direct influence on them. The primary goal of this academic exercise is to simulate how the original CPS, now with the newly implemented and specialised protective shell, would have handled the situation when faced with the same failures or exploitations that caused the accidents or incidents.
Furthermore, the effectiveness of each specific protective shell will be evaluated based on the vulnerabilities that could have been mitigated during run-time and whether the catastrophic outcomes could have been prevented or their consequences drastically reduced. The evaluation will also consider a broader perspective on the CPS's attack surface and fault modes to determine whether the implementation of a protective shell has increased the number of attack vectors or failure modes. It is essential to be cautious while implementing mitigating techniques to reduce residual risk, as the system's complexity increases, potentially introducing new weaknesses and vulnerabilities.
Finally, the similarities among the different specific protective shells will be summarised, leading to valuable lessons learned and the identification of *core characteristics for protective shells*, holding high significance for the subsequent chapters.

5.1 Boeing 737 Max MCAS

5.1.1 Introduction

The Boeing 737 Max MCAS safety accident occurred in 2018 and 2019, involving two fatal crashes of Boeing 737 Max aircraft. The *Manoeuvring Characteristics Augmentation System* (MCAS) was a newly implemented feature on the 737 Max, designed to prevent the aircraft from stalling. However, the system relied on data from two angle of attack (AOA) sensors that were not properly validated, resulting in inconsistent readings. A single sensor providing incorrect data was sufficient for the MCAS to repeatedly push the aircraft's nose down, overriding the pilot's manual interventions, ultimately leading to the crashes. The safety issue initially came to light following the crash of Lion Air Flight 610 in Indonesia in October 2018, claiming the lives of all 189 passengers and crew members on board. A few months later, in March 2019, Ethiopian Airlines Flight 302 crashed, killing all 157 people on board.
The investigations revealed that Boeing had not adequately informed pilots of the MCAS system and how to override it in case of malfunction. Additionally, it was found that the FAA (Federal Aviation Administration) had not been provided adequate oversight of the certification process for the MCAS system. As a result of the accidents, the Boeing 737 Max was grounded worldwide for almost two years, and Boeing faced significant financial and reputational losses. The incident also led to increased scrutiny of the aviation industry's safety practices and regulatory processes.

5.1.2 Vulnerabilities within CPS

The Federal Aviation Administration (FAA) identified seven key safety concerns associated with the Boeing 737 Max crashes [16]. Presented below is a summary highlighting the most significant issues.

Single point of Failure The most severe safety issue that contributed to the Boeing 737 Max crashes was the single-point-of-failure vulnerability in the MCAS system. The MCAS system used two angle sensors, located on the left and right sides of the cockpit. It was anticipated that both sensors would provide similar readings for the MCAS to operate effectively. However, this expectation was not enforced through the validation of sensor readings to ensure their consistency and reliability.
If one of the sensors failed, it would transmit erroneous data to the flight control computer. Consequently, the flight control computer would activate the MCAS stabiliser, resulting in a nose-down command. This meant that the initiation of the nose-down manoeuvre relied solely on the readings from one sensor and did not require confirmation from other sensor inputs. Moreover, there was no safeguard in place to respond appropriately if the AOA sensor readings indicated the presence of false data.
Unless the flight crew promptly disabled the stabiliser trim motor, including the MCAS, the repeated execution of the MCAS-induced nose-down command would lead to a rapid loss of altitude, acceleration in speed, and ultimately result in a crash. This scenario is a typical example of a single point of failure, where the failure of a single component has cascading effects on the entire system, ultimately leading to an accident.

Repetitive Execution of MCAS Commands The FAA discovered that the MCAS activation was reset too quickly following the initial nose-down command, allowing the flight control computer to reactivate it and initiate repetitive nose-down commands based on the inaccurate sensor readings. It was observed that each time the pilot issued an electric trim command and released the trim switch, the MCAS activation was reset [16]. A correct implementation of MCAS, designed to prevent the aircraft from stalling after executing the nose-down command, would wait for the aircraft's angle to stabilise before allowing for any subsequent MCAS activation.

Static Nose-Down Command When the nose-down command is executed, it causes a fixed movement of the horizontal stabiliser, regardless of its current position. The repetitive execution of the nose-down command results in incremental adjustments of the horizontal stabiliser, with each adjustment adding a fixed amount. It is crucial for the MCAS system to consider the current position of the horizontal stabiliser before making any adjustments, ensuring that it remains within safe operational limits.

Insufficient Documentation and Training One significant safety issue with MCAS was the lack of sufficient documentation and training provided to the flight crews. Following the accidents, it became apparent that the critical automated system, MCAS, was not adequately explained in the flight manuals and training materials. As a result, pilots were largely unaware of its existence and its potential impact on aircraft control. Insufficient information and guidance on how to respond to erroneous sensor inputs and MCAS activations contributed to the challenges faced by the flight crews during emergencies. Furthermore, the alert message if the AOA sensor readings disagree was not visible by default, making it more difficult for the crew to find the root cause of the error. This highlighted the importance of

thorough documentation and comprehensive training programs to ensure the safe operation of complex automated systems in modern aircraft.

5.1.3 Specific Protective Shell Mitigation Mechanisms

The specific protective shell developed for the Boeing 737 Max MCAS incorporates several core mitigation mechanisms to address the faults that led to the accidents previously analysed. In the following section, those mitigation mechanisms are presented and build the main characteristics of the protective shell.

Reevaluation of Sensor Readings The protective shell can access all sensor readings passing through it, including the AOA sensors. This enables the shell to perform various checks on the sensor data, such as *cross-validation and plausibility checks*. If one of the two AOA sensors delivers erroneous readings, the simplest way of identifying such failure is to check if the two AOA sensor readings deviate from each other. If a significant deviation is detected, it would give a strong indication that the readings could not be trusted, prompting the protective shell to take appropriate actions, such as deactivating the MCAS system, to prevent cascading effects.
Furthermore, the protective shell can cross-validate the sensor readings with additional information. Domain experts have defined the desired behaviour using rule-sets, policies, and functional specifications, which should include intrinsic information about the correlation between the AOA sensor, barometer, and speedometer (e.g., a simplified gradient triangle) as depicted in figure 5.1. The protective shell monitors the ground and/or flight velocity as well as the pressure change over time and can calculate the distance travelled as well as the height gained or lost, respectively. Although this presentation retains a simplification for illustrative purposes, neglecting the earth's curvature by restraining from using the haversine formula for distance calculation, the angle of attack can be estimated using those distances and the presumed orthogonal 90-degree angle. Therefore, the different sensors exhibit a direct correlation. If this correlation does not hold, it serves as a hint to the protective shell that there must be erroneous sensor readings within the data streams.

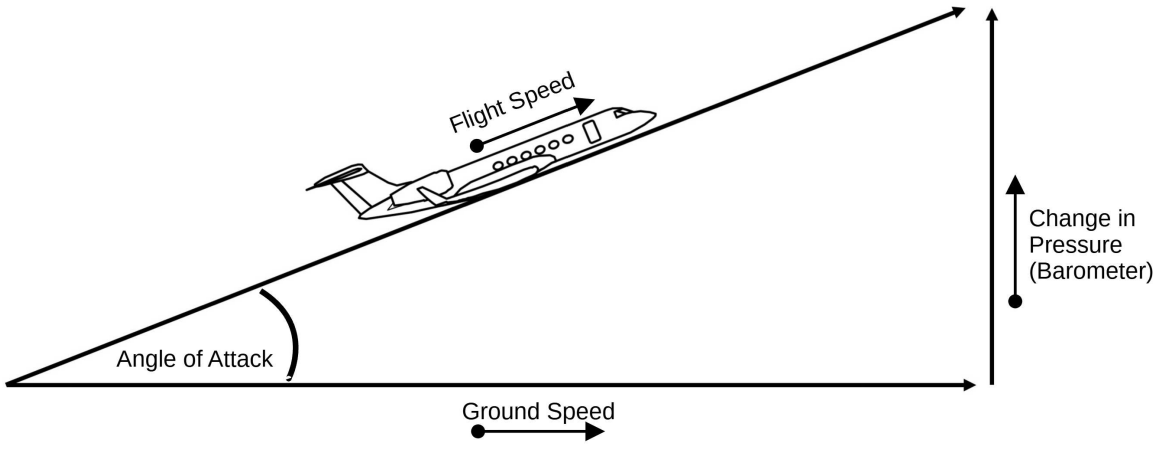


Figure 5.1: Boeing 737 Max - Climb Gradient Triangle [17]

Implementation of Supplementary Sensor To enhance redundancy and fault tolerance, the protective shell can implement an additional, independent AOA sensor to report to the protective shell. With three independent AOA sensors, the protective shell can employ the

"Triple Modular Redundancy" technique commonly used in safety-critical systems. This approach mitigates single points of failure and improves availability through voting capabilities. With two AOA sensors and one of them being defective, the defect can be detected, but without further investigation, it is unclear which sensor is functioning correctly.
The advantage of employing three sensors is that, in the event that one fails, the MCAS system can still receive a sensor reading that is trusted through either majority voting or average voting.

- Majority voting: This method forwards the sensor reading that occurs at least two times, eliminating the false reading of the failing sensor.
- Average voting: This method computes the average of all three sensor readings to be forwarded to the underlying system. It is useful when the three sensors exhibit different levels of accuracy or precision. Moreover, if one sensor reading varies significantly from the other two (beyond a predefined threshold), it can be excluded through outlier elimination to avoid falsifying the average.

By using three independent AOA sensors and employing appropriate voting mechanisms, the protective shell can significantly enhance the robustness and reliability of the system, reducing the likelihood of accidents caused by erroneous sensor data.

Range Validation One of the key mitigation mechanisms that can be incorporated into a protective shell for the Boeing 737 Max MCAS is range validation for sensors and actuators. The protective shell should define hard operation ranges for each sensor and actuator, ensuring that they operate within predefined limits. If any component operates outside these validated ranges, it would trigger an immediate warning to indicate a potential malfunction. For instance, in the case of the MCAS system, a range validation mechanism could prevent it from acting on false angle data. If the readings from the angle of attack (AOA) sensor fall outside the validated range, indicating a potential defect in the sensor, the protective shell would block these data streams from reaching the run-time core of the system.
Similarly, the protective shell should implement range validation for other critical actuators and sensors throughout the aircraft. For example, considering the horizontal stabiliser, the protective shell could enforce range validation to prevent extreme and unsafe adjustments. As analysed before, the horizontal stabiliser would be moved by a fixed amount when the nose-down command is executed. Through repetitive execution of the nose-down command, it is likely that the horizontal stabiliser was way outside its safe operation range. By blocking such extreme values from reaching the horizontal stabiliser, the protective shell would ensure that it remains within its safe operation range, giving the flight crew more time to respond effectively.

Time-Sequential Analysis and Retrospective Data Evaluation The specific protective shell designed for the Boeing 737 Max MCAS leverages time-sequential analysis, granting access to the current operational data and log files of the aircraft. By comparing this real-time data to predefined models and specifications describing the desired behaviour, the protective shell gains valuable insights into the system's performance. Additionally, the protective shell maintains a retrospective view on the data streams received previously, allowing for a comprehensive analysis of historical trends.
For instance, when dealing with sensor readings, the protective shell can contextualise individual readings by considering past values. This retrospective approach enables the detection of anomalies that may not be apparent from isolated data points. For example, if an AOA sensor experienced a failure after takeoff, the protective shell would recognise this abrupt breakdown by examining the time-sequential readings. Such drastic and sudden

changes in sensor data are unlikely in the physical world, and the protective shell can use this information to identify erroneous data. An AOA sensor reading that exhibits an instant 10-degree jump, for instance, would be flagged as an anomaly, given the implausibility of such a manoeuvre for a Boeing 737 Max passenger aircraft.
Moreover, the protective shell incorporates models and specifications defined by domain experts to recognise data trends and variations. Figure 5.2 depicts some typical trends and variations for sensors and actuators over a specific time-span. With this knowledge, the protective shell can establish expected patterns for various flight scenarios. For instance, the protective shell would anticipate that AOA sensors typically follow specific trends during flights. However, if a sensor diverges beyond a predefined threshold from the expected trend, the protective shell interprets it as an indication of an approaching failure mode.

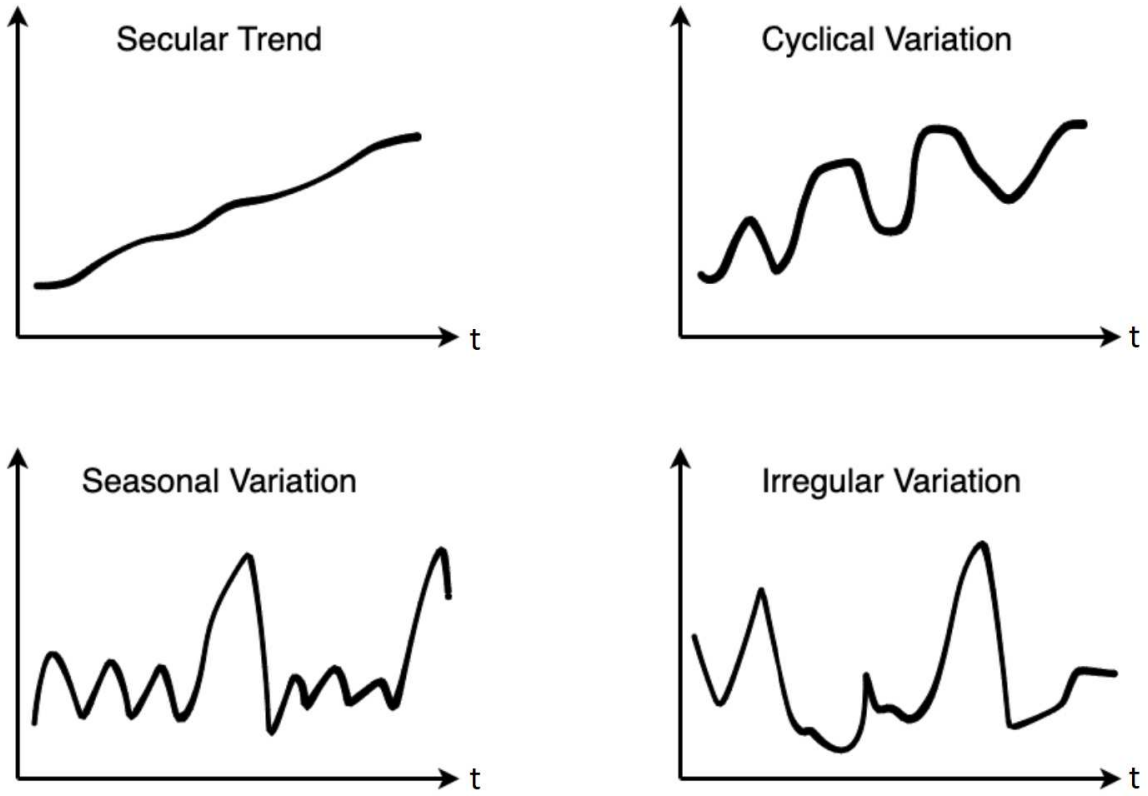


Figure 5.2: Boeing 737 Max - Typical Time Series Trends [18]

Block Repetitive Nose-Down Commands One of the key mitigation mechanisms implemented in the protective shell for the Boeing 737 Max MCAS is aimed at blocking repetitive nose-down commands. The protective shell will actively monitor the commands sent to actuators, for instance those sent to the horizontal stabiliser like the nose-down command initiated by the MCAS system. To prevent rapid and continuous nose-down adjustments, the protective shell will introduce a time-out mechanism. This time-out period will allow sufficient time for the aircraft to react to previous commands and stabilise before accepting new nose-down commands. By introducing a latency factor to account for the passenger aircraft response time and the delay in actuator control from the cyber component, the protective shell ensures that the aircraft first returned to a safe flight angle. Additional MCAS activation and consequently nose-down commands will only be permitted when the aircraft drifts back from a save angle to an angle where it could experience stalling.
This mechanism ensures that the aircraft's stabilisation is prioritised, reducing the risk of

rapid altitude loss, speed acceleration, and potential crashes caused by continuous nose-down adjustments.

Behaviour Model Training In addition to the previously discussed mitigation mechanisms, the specific protective shell for the Boeing 737 Max MCAS leverages behaviour model training to enhance its overall effectiveness. While the previous mechanisms were based on predefined functional specifications, policies, and rule-sets established by domain experts, they inherently have limitations as they primarily address known vulnerabilities. However, within this academic scenario, there exists the opportunity to develop a more precise and specialised desired behaviour, carefully tailored to address accidents triggered by MCAS.
Defining such a precise desired behaviour manually is challenging and time-consuming, making it impractical in real-world applications where unforeseen emergent properties may arise. To overcome this, the protective shell incorporates anomaly detection based on behaviour models that are automatically trained using appropriate information retrieval techniques.
To create the behaviour model, the protective shell requires a vast amounts of operational data and log files from various components of the Boeing 737 Max. These data-sets contain a wealth of intrinsic information, including correlations, patterns, rules, and time series, that can be challenging for human experts to extract comprehensively. Leveraging machine learning algorithms, the protective shell can analyse thousands of Boeing 737 Max flights to understand how the MCAS system should function ideally. This analysis results in the pre-calculation of a behaviour baseline, which serves as a reference for identifying deviations from the expected behaviour of the MCAS system.
By incorporating behaviour models, the protective shell gains a deeper understanding of the MCAS system's expected behaviour, allowing it to identify abnormal patterns or actions that could potentially lead to safety risks. This automated and data-driven approach significantly enhances the protective shell's ability to respond effectively to anomalies, even those that might not have been explicitly defined in the initial static guidelines provided by domain experts.
For a more comprehensive exploration of the behaviour model training and its impact on the protective shell's effectiveness, further details will be covered in chapter 6 of this thesis.

Alerting and Logging The protective shell developed for the Boeing 737 Max MCAS system incorporates several crucial mitigation mechanisms to enhance the overall safety and security of the CPS. One of the primary features is its alerting capabilities, which play a vital role in promptly notifying the CPS operator whenever anomalies are detected, leading to necessary interventions. This immediate alerting ensures that the operator is aware of potential dangerous behaviours the aircraft is experiencing or the failure mode it is heading to.
By alerting the operator in real-time with the anomalous findings, the protective shell grants them the opportunity to analyse the situation independently and, if necessary, take over manual control. This capability not only empowers the operator to take corrective actions but also puts pressure on the manufacturer, Boeing, to investigate the cause of the alarm. In the event of unexpected negative behaviour being identified, appropriate mitigation measures can be put in place promptly. Alternatively, if the alert turns out to be a false alarm, the protective shell can be adjusted accordingly, preventing unnecessary disruptions.
It is crucial that the protective shell maintains a comprehensive log of all instances where it passively alarms the operator or actively executes protective actions. Additionally, when the protective shell monitors the CPS without detecting any anomalies, logging should be considered, although the volume of data generated must be balanced to remain practical. Extensive logging is particularly critical when anomalies are detected and protective actions

are initiated, as this data will provide engineers with valuable forensic insights into the CPS's behaviour and help identify the reasons behind the protective shell's actions.
Furthermore, detailed logs are essential in case of incidents where the protective shell's actions might have resulted in fatalities, injuries, or property damage. Having access to comprehensive logs will enable the manufacturer to analyse the sequence of events, understand why the protective shell made specific decisions, and offer explanations in case of any unintended consequences.
The alerting capabilities of the protective shell must be seamlessly integrated into the operator's user interface to ensure timely and effective communication. Alarms should be displayed prominently and clearly, avoiding any overlap with other screen elements. The alert should not be automatically hidden or obscured like it was implemented in the MCAS system where the alert that the AOA sensors disagree was not visible by default, as this defeats the purpose of the warning. While the operator may manually suppress the alert if it turns out to be a false alarm, the protective shell should prioritise displaying critical notifications noticeably to avoid oversight and ensure quick response in critical situations.
Overall, the specific protective shell developed for the Boeing 737 Max MCAS system incorporates alerting capabilities and extensive logging to enhance safety, provide necessary information for further analysis, and empower operators to respond effectively to anomalies in real-time.

5.1.4 Protective Shell Evaluation

This section evaluates whether the two fatal safety accidents involving Boeing 737 Max aircraft could have been prevented if the specific protective shell, equipped with the presented mitigation mechanisms, had been implemented. The protective shell for the Boeing 737 Max MCAS aims to address vulnerabilities identified in the original MCAS, particularly the single-point-of-failure issue associated with the Angle of Attack (AOA) sensors, which led to incorrect nose-down commands and fatal accidents. This evaluation examines if the protective shell's features effectively mitigate these single-point-of-failures through various monitoring and validation processes, and if new vulnerabilities got introduced.

Remediation of Old Failure Modes The original MCAS design featured two AOA sensors contributing to a fail-safe design through redundancy, capable of detecting a defective sensor if employed correctly. However, flawed implementation allowed one erroneous reading to cause the entire failure. Therefore, the first mitigation mechanism contained in the protective shell was to remedy the ineffective implementation done by the engineers, to truly contribute to a fail-safe design by first checking if the two sensor readings deviations exceed measurement inaccuracies, and if so block the erroneous data from being forwarded to the MCAS.
Another key mechanism is cross-validating data from the two AOA sensors against other correlated data streams, enabling the protective shell to identify and discard erroneous readings.
The implementation of a third independent AOA sensor adds redundancy and fault tolerance, utilising voting mechanisms to ensure reliable sensor data.
Furthermore, the protective shell enforces range validation for critical sensors and actuators, ensuring they operate within predefined safe limits.
Lastly, the protective shell also employs time-sequential analysis and retrospective data evaluation, comparing sensor readings with trained data trends and reevaluating current data points against previously sent data sets. This enables the detection of anomalies when an AOA sensor abruptly malfunctions or gradually becomes unreliable over time.
Therefore, the MCAS is now safeguarded through several mitigation mechanisms from AOA sensor failure, reducing the associated risk of AOA sensor failure to an acceptable level.

Nevertheless, assume the AOA failure could spread to the MCAS despite safeguards or MCAS would experience a fault exposed by another vulnerability. The specific protective shell suggests mitigation mechanisms that would isolate a malfunctioning MCAS, preventing cascading negative effects. If the MCAS initiates manoeuvre actions like nose-down commands repetitively deviating from the trained behaviour, the protective shell would intervene and block it from reaching actuators like the horizontal stabiliser and transfer control to the pilots. This is because the protective shell assumes that if it allows MCAS to act outside the behaviour model, the aircraft could drift into a failure mode that might lead to an accident.

Lastly, the protective shell features alerting and logging capabilities, providing pilots with valuable insights into detected anomalies through alarms, allowing them to initiate manual interventions. Furthermore, the detailed logs provide Boeing, the manufacturer, with valuable insights into CPS's behaviour, aiding in further analysis and understanding the reasons behind the protective shell's actions.

Identification of New Vulnerabilities While considering new vulnerabilities introduced by the protective shell, it is essential to analyse potential risks introduced by the mitigation

mechanisms. In the case of MCAS, the protective shell's mechanisms could lead to angle readings being discarded, potentially hindering the initiation of correct manoeuvres. In an unlikely event, this could result in the aircraft stalling due to a high angle of attack and experiencing a loss of lift, leading to a fall. However, pilots can counteract this situation by initiating recovery procedures, such as reducing the angle of attack and increasing speed.
It is crucial to note that the probability of such an event is very low because the protective shell employs cross-validation and other techniques to block sensor readings only when there is high certainty of their inaccuracy. Additionally, pilots are promptly made aware of any blocking actions taken by the protective shell, allowing them to intervene if necessary.

Conclusion The specific protective shell developed for the Boeing 737 Max MCAS effectively addresses the vulnerabilities present in the original MCAS design. By employing various mitigation mechanisms, the protective shell significantly enhances the trustworthiness of the CPS. However, the protective shell has no influence on the insufficient documentation and training provided by Boeing.
With the implementation of this protective shell, it is plausible that the two fatal safety accidents involving Boeing 737 Max aircraft could have been prevented, potentially saving numerous lives and avoiding the subsequent worldwide grounding of the aircraft. The thorough evaluation of the protective shell's features and their impact on avoiding failure modes demonstrates the significance of such safety measures in cyber-physical systems, particularly in critical domains like aviation.

5.2 Therac-25

5.2.1 Introduction

The Therac-25 medical device was involved in one of the first documented safety accidents involving CPS in the mid-1980s. The Therac-25 was a radiation therapy machine utilised for treating cancer patients. It was designed to deliver high doses of radiation to tumours while minimising exposure to healthy tissue. Unfortunately, a software design flaw in the control system led to massive overdoses of radiation to at least six patients, resulting in severe injuries and fatalities.
The software flaw was specifically associated with the system's user interface, which had been designed to enhance user experience compared to its previous version. The user interface allowed operators to manually input treatment parameters, including the type and dosage of radiation to be delivered. However, a race condition within the software existed, whereby if the operator entered certain parameters too quickly, the system would not check them for consistency before proceeding with the treatment. Consequently, several incidents occurred where patients were exposed to radiation doses significantly higher than intended [19].
This incident underscored the critical importance of incorporating safety design principles, such as system interlocks, during the development of safety-critical systems.

5.2.2 Vulnerabilities within CPS

No Hardware Interlocks The engineers at Atomic Energy of Canada Limited (AECL) incorporated pre-existing in-house software from the Therac-6 and Therac-20 into the Therac-25, introducing electron-beam therapy and megavolt x-ray functionality simultaneously. Unfortunately, the legacy software contained bugs that were inherited by the Therac-25 software. Unlike its predecessors, the older radiation therapy machines relied on hardware interlocks to validate crucial safety parameters before starting treatment. These interlocks concealed the software defects, but did not issue any warnings to the system's operator. A significant safety concern with the Therac-25 was the absence of these hardware interlocks, resulting in the electron-beam operating in high-energy mode even when the required target was not in place [19]. The engineers made the decision to rely solely on software checks to ensure safe operation and neglected to implement an interface for the hardware components to verify they are operating as intended.

Software Bugs As previously mentioned, the Therac-25 contained several software bugs that resulted in radiation overdoses for the patients. One of these bugs stemmed from a race condition during the configuration of parameters. When the doctor mistakenly selected x-ray mode and quickly switched back to electron mode, the electron-beam was erroneously set with a radiation dose approximately 100 times higher than intended [20].
A second issue arose from a software bug during the field-light mode. The intended purpose of the field-light mode was to only illuminate the specific area of radiation which was configured, for refined adjustments, directly on the patient. However, the software bug caused the electron-beam to activate during this pre-check mode, leading to radiation burns, poisoning, and three deaths as a consequence of the overdose.
The final significant software bug which will be discussed in this paragraph involves an arithmetic overflow within one of the system's flag variables. Those flag variables play an important role in essential safety checks prior to starting the treatment. However, this flag variable progressively increased in value after each operation until it exceeded the supported range, thereby bypassing crucial safety checks [19].

No Thorough Testing Insufficient testing, although not inherently a vulnerability, opens the door for software bugs. It is crucial to highlight the shortcuts taken by the engineers at AECL during the development process, which compromised safety and security requirements.
As previously mentioned, AECL utilised in-house code from the Therac-6 and Therac-20 as the foundation for the Therac-25 software and operating system. The trust placed in this in-house code was so strong that it was deemed unnecessary to have an independent entity review it. The engineers held a collective belief that the software was free of bugs. This overconfidence led AECL to assure their customers that overdoses caused by the Therac-25 were impossible, slowing down the thorough analysis of post-accident issues [21].
Remarkably, the Therac-25's development process entirely neglected a risk management procedure such as the failure mode and effect analysis (FMEA). The engineers should have diligently followed this safety risk assessment methodology to identify potential failure modes, assess their impact, and, if possible, implement mitigation measures prior to their occurrence during system operation. Furthermore, it became evident that AECL did not conduct thorough testing of the entire CPS. Instead, they sporadically tested the software and hardware separately. Consequently, the testing of the complete assembled CPS began at the customers' hospitals, involving real patients, rather than being conducted beforehand.

5.2.3 Specific Protective Shell Mitigation Mechanisms

The specific protective shell developed for the Therac-25 incorporates several core mitigation mechanisms to address the vulnerabilities that led to the accidents previously analysed. In the following section, those mitigation mechanisms are presented, which build the main characteristic of this protective shell.

Integrity Validation for Parameters One crucial mitigation mechanism implemented in the protective shell is the integration of back-end validation for parameters configured through the graphical user interface (GUI) by the operator. The desired behaviour of the Therac-25 is established by domain experts through well-defined specifications, rules, and policies, which outlines the radiation dose for different modes, such as x-ray, electron, and field-light. Depending on the selected mode, the protective shell validates whether the radiation dose falls within safe operating ranges, establishing a restricted operation envelope (like used in the aviation domain with its safe flight envelope). Notably, the radiation dose in x-ray mode is significantly higher than in electron mode, and no radiation dose is allowed in field-light mode.

One crucial mitigation mechanism implemented in the protective shell is the integration of back-end validation for parameters configured through the graphical user interface (GUI) by the operator. The desired behaviour of the Therac-25 is established by domain experts through well-defined specifications, rules, and policies, which outlines the radiation dose for different modes, such as x-ray, electron, and field-light. Depending on the selected mode, the protective shell validates whether the radiation dose falls within safe operating ranges. Notably, the radiation dose in x-ray mode is significantly higher than in electron mode, and no radiation dose is allowed at all in field-light mode.
If the parameters set do not match the rules specified, the protective shell will block the treatment to start, discarding the application signals send to the actuators to proceed with the treatment.
The implementation of this validation mechanism effectively prevents race conditions during rapid mode switching by the operator. Additionally, the protective shell incorporates crucial safety checks before the treatment starts already existing in the Therac-25 software, providing redundancy. This addresses the vulnerability in the Therac-25 where safety checks were

bypassed due to an arithmetic overflow of an important flag variable. By reexamining safety checks within the protective shell, this vulnerability is mitigated.
Consequently, this added layer of protection ensures that erroneous or faulty parameters are not accepted by the Therac-25. If the protective shell identifies any concerns regarding the correctness of provided parameters, it blocks the treatment initiation, safeguarding against potential accidents.

Supplementary Sensor and Actuator Data An essential enhancement introduced in the protective shell involves the inclusion of hardware interlocks, which were lacking in the original Therac-25 design. With engineers neglecting to implement hardware interlocks and the decision to solely rely on software checks with vulnerabilities present in the application, it made the safety accidents possible. By implementing previous suggested mitigation mechanisms to reevaluate parameters provided by the operator, the software checks should now be effective. However, these interlocks play a crucial role in ensuring fault tolerance and further fortifying the safety measures. To implement hardware interlocks, the protective shell requires access to additional data streams, including operational data from various components and supplementary sensors.
One valuable hardware interlock is the mechanical position verification. It consists of supplementary sensors, such as limit switches and proximity sensors, to detect the presence and position of crucial components before the treatment starts. For instance, before enabling the electron-beam in high-energy mode, the interlock would verify if the required target is correctly in place and properly aligned.
Another vital hardware interlock is the radiation monitoring system. This system continuously measures the intensity and direction of the radiation beam. If any deviation from the expected radiation levels or direction is detected, the hardware interlock automatically halts the treatment, preventing potential harm to the patient.

Alerting and Logging Similarly to the mitigation mechanisms presented in the Boeing 737 Max, the Therac-25 also needs alerting and logging capabilities for likewise reasons. These features serve crucial purposes in ensuring transparency and aiding operators in understanding and responding to potential issues promptly.
If the protective shell blocks treatment initiation, an alarm must be displayed on the operator's GUI, providing a clear explanation of the reason for the treatment's suspension. Additionally, in cases where hardware interlocks suspend an ongoing treatment due to deteriorating operational conditions, the operator is immediately informed about the detected issues and provided with relevant guidance on resolving them.
Furthermore, the protective shell maintains extensive logs whenever it detects anomalies and intervenes. These logs serve as valuable data for the manufacturer to conduct thorough analyses. They facilitate the identification of underlying issues within the Therac-25 architecture and contribute to refining the protective shell's behaviour, especially if the detection of anomalies was erroneous.

5.2.4 Protective Shell Evaluation

This section critically assesses whether the severe injuries and deaths associated with the safety accidents of the Therac-25 medical device could have been prevented through the implementation of a specific protective shell, equipped with previously presented mitigation mechanisms. The primary objective of the protective shell is to address vulnerabilities identified in the original CPS architecture of the Therac-25.
This evaluation aims to determine the effectiveness of the protective shell's capabilities in mitigating identified vulnerabilities through various monitoring and validation processes, thereby reducing residual risk. Additionally, it investigates whether the implementation of the protective shell resulted in new vulnerabilities.

Remediation of Old Failure Modes The original Therac-25 architecture lacked crucial hardware interlocks, and its flawed software implementation allowed treatments to be initiated with dangerous parameters. Therefore, the first important mitigation mechanism incorporated in the specific protective shell was to rectify the flawed software implementation and ensure proper validation of treatment parameters. Any parameters not adhering to the policies and rule-sets specified by domain experts in the protective shell's desired behaviour are promptly deemed invalid, thereby preventing potential harm to patients and blocking the treatment from starting.
Furthermore, safety checks implemented in the application were replicated in the protective shell to effectively adhere to the hard rules. This duplication significantly reduces the likelihood of bypassing these safety checks due to arithmetic overflows or other software flaws, providing an additional layer of protection.

Another key mitigation mechanisms is the implementation of hardware interlocks, as presented previously. These interlocks act as a fail-safe measure, intervening if hazardous operation conditions persist despite software-based safety checks. For instance, hardware interlocks can promptly interrupt the treatment if an incorrect radiation dose is detected or if the target in electron beam mode is misplaced.

Lastly, the protective shell features alerting and logging capabilities, offering doctors valuable insights on their GUI regarding detected anomalies or invalid parameters, guiding them how to address the issues. Additionally, detailed logs provide the manufacturer, AECL, with valuable information about the CPS's behaviour, aiding in further analysis and understanding of the protective shell's actions.

Identification of New Vulnerabilities Since the mitigation techniques employed in this specific protective shell imposes more stringent operational parameters compared to the original Therac-25, no new failure modes or attack vectors are introduced. However, it is essential to acknowledge that these stricter specifications and hardware interlocks might result in an increased number of false-negatives, leading to reduced recall (also known as sensitivity). Nevertheless, this decrease in recall should not be considered critical, as the treatment parameters can be validated by the medical staff and restarted. The primary concern is preventing dangerous treatments from being initiated, making the precision value more safety-critical than its recall counterpart.

Conclusion The protective shell designed for the Therac-25 incorporates multiple mitigation mechanisms, significantly enhancing the system's trustworthiness. It effectively addresses the vulnerabilities present in the original Therac-25 design. By validating parameters, implementing hardware interlocks, and providing alerting and logging functionalities, the protective shell effectively reduces the risk of accidents and ensures a higher level of confidence in the system's operation.

It is highly plausible that the injuries and fatalities resulting from the Therac-25 safety accidents could have been prevented with the implementation of this protective shell. However, it is essential to note that the protective shell cannot rectify issues arising from insufficient testing during the development process or miscommunication by AECL, where they wrongly assured customers that radiation poisoning could not be attributed to their Therac-25 product.

The thorough evaluation of the protective shell's features and their impact on avoiding failure modes demonstrates once again the significance of such safety measures in cyber-physical systems, also in critical domains like medical applications.

5.3 Stuxnet

5.3.1 Introduction

The Stuxnet cyberattack is widely acknowledged as one of the most sophisticated incidents in the history of cyber warfare. It was discovered in 2010 and is believed to have undergone several years of development prior to its identification. This attack specifically targeted Iran's nuclear facilities by exploiting vulnerabilities within Microsoft Windows operating systems and Siemens software used in industrial control systems. The malware successfully infiltrated and manipulated programmable logic controllers (PLCs) used in uranium-enriching centrifuges. Upon payload delivery, the malware caused the centrifuges to spin out of control and subsequently self-destruct.
Quantifying the extent of damage inflicted by Stuxnet is a challenging task, as much of the information surrounding the attack is classified. However, it is widely speculated that this malicious software caused significant harm to Iran's nuclear program, resulting in a considerable setback and operational disruptions. While it is commonly believed that the United States and Israel collaborated on this cyber offensive against Iran, neither nation has officially acknowledged their involvement in the attack.

5.3.2 Exploited Vulnerabilities

The Stuxnet worm successfully bypassed numerous obstacles by exploiting multiple vulnerabilities, ultimately achieving its goal of setting back Iran's nuclear program.

Initial Infection The nuclear facility had stringent physical and network restrictions. Only authorised personnel were permitted to enter, and access to the internet from the internal network was mostly disabled. These conditions can be described by the term "Air-Gap" - an insulating layer that separates everything behind it [22]. Therefore, it was assumed to be near impossible to launch a successful cyberattack on the nuclear facility.
However, as mentioned earlier, Stuxnet managed to infect the internal computer network of the facility. It is widely believed that Stuxnet crossed the Air-Gap by means of an infected USB stick. Once an employee connected such infected USB stick to an internal computer within the facility it automatically spread to the computer, infecting it also.
This highlights several vulnerabilities that allowed the worm to bypass the insulating layer.
Firstly, there was human error involved, as the same USB stick was used across different levels of security zones. By connecting this drive, which could potentially contain malware, to an internal computer, it exposed the internal network to external threats.
Secondly, the internal computer seemed to not be hardened enough against such threats. It is common practice to disable USB ports or remove them completely to prevent this way of initial infection in the first place.
Thirdly, the physical access control at the facility did not effectively prohibit employees from bringing in external hard drives. Either it was not explicitly forbidden or the enforcement was not thorough enough.

Privilege Escalation Once Stuxnet successfully infected the victim's computer, it needed to acquire administrator privileges in order to fully leverage its capabilities. To achieve this, it utilised one of two privilege escalation zero-day exploits for the Windows 32-Bit operating system. The term "zero-day" refers to a software vulnerability that the software vendor was unaware of at the time of exploitation. This means that the software company had zero days to mitigate the risk, and only the discoverer of the vulnerability could exploit it.

The first available zero-day exploit in Stuxnet for gaining elevated privileges targeted a vulnerability in the Windows task scheduler (CVE-2010-3338). The task scheduler is a common feature in operating systems that enables users to schedule the execution of specific programs at designated times and under specified security contexts. The Windows task scheduler had certain flaws that allowed a standard user without administrator privileges to run an arbitrary program with system permissions. To make this work, the user had to manipulate the task in a certain manner during its creation. By scheduling a program that modifies the permissions for the local user and executing it with elevated permissions, the user successfully and persistently escalated their privileges.
The second privilege escalation zero-day in Stuxnet's arsenal involved exploiting a vulnerability in the Windows 32-Bit kernel (win32k.sys). A local user could flip a Bit at an arbitrary kernel address through a particular system call. This call was mishandled by the Windows kernel, resulting in the user elevating their permissions to system [23].

Infecting new computer One of the key characteristics of computer worms is that they replicate themselves to infect as many computers as possible. By infecting a larger number of computers, the probability of successfully delivering the payload, thereby achieving an overall successful attack, increases. In the case of Stuxnet, the goal was to navigate through the network until reaching the Siemens PLCs and reprogram them. Consequently, Stuxnet was equipped with several powerful tools to spread to new computers while minimising the chances of being trapped in a dead end.
The first tool utilised by Stuxnet to gain a foothold within the facility was the .LNK/.PIF zero-day exploit (CVE-2010-2568). By specifically crafting .LNK/.PIF shortcuts, the engineers behind Stuxnet were able to execute arbitrary code on the victim's machine simply by the local user opening a folder containing such a shortcut. Therefore, when the infected USB stick was connected to a PC and the user opened the folder, their machine became infected. Furthermore, Stuxnet also spread to new, uninfected drives that were connected to an infected machine by creating these invisible shortcuts [24].
The second method of spreading to new computers involved the internal network connection. If network drives were accessible, Stuxnet attempted to copy itself onto those drives and infect any machine accessing them. Another option for infecting computers on the local network was through remote procedure calls, along with trying different user credentials.
Lastly, the most effective way for Stuxnet to spread through the network was by exploiting another zero-day vulnerability known as the Print Spooler Service Impersonation Vulnerability (CVE-2010-2729). This vulnerability is particularly severe as it allows the attacker to remotely execute arbitrary code on the victim's machine. To exploit this vulnerability, printer sharing needed to be enabled on the target machine. The attacker could then send a manipulated print request, bypass spooler access permission validation, and ultimately create files in the system directory [22].

Payload Delivery As previously mentioned, the ultimate objective of Stuxnet was to hinder and delay Iran's nuclear program. To accomplish this, the attackers aimed to sabotage the uranium-enriching centrifuges in the nuclear facility. These centrifuges were controlled by programmable logic controllers (PLCs), specifically the Siemens Simatic-S7. If Stuxnet managed to propagate deep into the facility's local network and reach a computer connected to the S7, it would be possible to reprogram the PLCs by redirecting, altering, or intercepting calls to the centrifuges. However, simply reprogramming the PLCs would be an easily reversible task, leading only to the destruction of the current batch of enriched uranium, which would likely be noticed by personnel. Considering the significant monetary and time investment required to develop such a sophisticated malware, it seems unlikely that the temporary delay of a few weeks in the nuclear program would justify the effort. Consequently,

the reprogramming of the PLCs had to cause more severe and lasting damage to the facility. This highlights the first major vulnerability in the combination of Siemens Simatic-S7 and the centrifuges.
The centrifuges have a designated safe operating frequency range between 807 Hz and 1210 Hz [22]. The CPS should have validated and enforced this range to prevent the centrifuges from operating outside the norm. However, this validation did not occur, allowing the attacker to send instructions for the centrifuges to spin at high frequencies (1410 Hz) initially and later at low frequencies (2 Hz). During the high-frequency operation, the components of the centrifuges experienced accelerated degradation and increased failure rates over time. After subjecting the centrifuges to wear, the sudden reduction in operating frequency to 2 Hz, as instructed by Stuxnet, led to desynchronization and significant misalignment between the rotor and casing. This imbalance disrupted the smooth and efficient operation of the centrifuges, further accelerating their deterioration. Eventually, the bearings failed, causing irreparable damage to the centrifuges.
The second major vulnerability in this CPS was the personnel's inability to accurately monitor the centrifuges with their programs. The degradation process initiated by Stuxnet was not rapid, but rather a gradual one that likely took several months before noticeable effects appeared in the centrifuge components. Therefore, it was crucial for the success of the attack to delay detection for as long as possible. Any deviation in the rotation speed of the centrifuges outside the frequency range predefined by the personnel, would be detected rather quickly. Hence, when Stuxnet reached the Simatic-S7, it did not actively start the attack, rather passively monitored the operating frequency of the centrifuges for approximately 13 days before taking over control. Once Stuxnet started modifying the rotation frequencies of the centrifuges, it would simultaneously transmit false rotation speeds previously collected back to the personnel's monitoring systems, signalling that everything was functioning normally [24]. The underlying vulnerability exploited in this case was a flaw in the system's design. The integrity of the centrifuges' sensor data must be protected, and any tampering should be immediately detected. Additionally, the monitoring system should retrieve input data directly from the sensor source without any potential for malicious modifications.

5.3.3 Specific Protective Shell Mitigation Mechanisms

The specific protective shell developed for uranium-enriching centrifuges incorporates various core mitigation mechanisms to address the vulnerabilities discovered in the previously analysed security incident. The purpose of this protective shell is to safeguard the CPS, consisting out of the centrifuges and the programmable logic controllers. Consequently, the protective shell does not directly contribute to the security of the Windows computer system that Stuxnet exploited to spread to the operational technology (OT) network.
In the following section, the mitigation mechanisms are presented that constitute the main characteristics of this protective shell.

Range Validation The protective shell relies significantly on the implementation of range validation for sensors and actuators reporting to the Siemens Simatic-S7 system. This crucial mitigation mechanism involves defining strict measurement and operation ranges for these components, respectively. The ranges for sensors are established based on physical laws or specifications provided by the sensor manufacturers. Simultaneously, for actuators, hard operational constraints are introduced, ensuring their operation remains within predefined limits.
The process of establishing these validated ranges involves the expertise of domain experts, who define functional specifications, rules, and policies that accurately reflect the capabili-

ties of the components and the requirements they must fulfil. By doing so, the protective shell can effectively monitor and enforce the permissible ranges, safeguarding the integrity of the CPS.
For instance, in the case of uranium-enriching centrifuges, the desired behaviour dictates a safe operation frequency range of 807 Hz to 1210 Hz. Consequently, the protective shell would diligently prevent any operation that falls outside this range, effectively mitigating potential risks associated with excessive degradation of the components. By adhering to the specified validated ranges, the protective shell ensures that the centrifuges operate within their safe and efficient operational parameters, reducing the likelihood of unexpected failures or unauthorised activities.
Furthermore, by establishing clear limits through range validation, the protective shell reinforces the stability and reliability of the system, thus enhancing the overall safety and security of the uranium-enriching process. This comprehensive approach plays a critical role in mitigating potential threats and faults, as it enables the protective shell to act proactively and prevent any deviations from the expected behaviour of the components, thereby protecting it from potentially harmful operations.

Behaviour Model Training Ensuring the trustworthiness of the centrifuges controlled by Siemens PLCs can be achieved through continuous comparison with a precalculated behaviour baseline. This behaviour baseline can be created best automatically by analysing a vast amount of operational data and log files from similar centrifuges that have successfully enriched uranium. Since this process is rather strict, the underlying data records would exhibit highly similar timelines and characteristics. Therefore, it simplifies the process of creating a behaviour baseline and makes it uncomplicated to identify anomalies by centrifuges that slightly deviate from their normal procedure.
This not only helps detect attacks similar to Stuxnet with great deviations but also aids in recognising subtle variations resulting from component degradation or more concealed malicious intent.
The Stuxnet attacker explored potential methods to disrupt the uranium enrichment process further by targeting batches of uranium hexafluoride gas. As Iran had a limited supply of this gas, its destruction could have significantly set back the nuclear program. While the destruction of centrifuges would have a more profound impact on the program, it was uncertain whether such an approach was feasible with Stuxnet at that time. By manipulating the operation frequency of the centrifuges and only deviating it a bit from the normal procedure (within safe operation frequencies), the attacker could destroy the current batch of gas. These subtle variations in operation frequency could be challenging to detect through conventional range validation mechanisms, or even by human supervisors monitoring the centrifuges. Therefore, the implementation of effective anomaly detection is important.
Another use case of employing anomaly detection through an automatically trained behaviour model, besides malicious modification of operation parameters, is predictive maintenance. Over time, components may experience slight and hard noticeable changes in their operation behaviour, which could compromise product quality and potentially disrupt critical processes, such as present in a nuclear program. Implementing predictive maintenance helps by timely repairing systems before they experience faults and stopping cascading negative effects on surrounding components. In time-critical processes, such as a nuclear program, unpredicted maintenance could set back the program considerably. Predictive maintenance detection is a great opportunity to be implemented in the protective shell. In this case it can prevent damage to property, but in other safety-critical CPS it might protect human lives.

Monitor Data Protection To counteract sophisticated attacks like Stuxnet, robust mechanisms must be implemented to protect the operational data transmitted to personnel monitoring systems. Stuxnet was particularly insidious as it initially recorded normal operational data from the centrifuges and later sent back tampered data to deceive personnel, making them believe that everything was operating as expected while the attack was underway and rotation frequencies were being manipulated.
Two essential mitigation techniques work collaboratively to reduce the effectiveness of such tampering:
1. Operational data sent to personnel monitoring systems should be signed using asymmetric cryptography to ensure data integrity and authenticity. In the event of a man-in-the-middle attack, where an attacker intercepts the connection between the CPS and the monitoring system, the signed data remains secure. The attacker cannot tamper with the data since any alteration would invalidate the signature, and the personnel monitoring system can verify the authenticity of the data using the associated certificates. This cryptographic approach safeguards the operational data from outside attackers by assuming an insecure connection between CPS and monitoring system.
2. However, in cases like Stuxnet, where the worm infects the CPS itself, cryptographic measures alone might prove inadequate if the attacker gains access to the certificate within the CPS's run-time core or can send out fake operation data with valid signatures on his own. To address this concern, the second mitigation mechanism comes into play. The responsibility for reporting operational data lies with the protective shell. The protective shell acts as an intermediary between the CPS core and the personnel monitoring system. It receives sensor readings, signs them using the cryptographic keys, and forwards them to the monitoring system before the potentially compromised CPS core can tamper with or discard the data. By effectively blocking and discarding any invalid operational data sent by the infected CPS core, the protective shell ensures that only accurate and trustworthy data is transmitted to the monitoring system. The firmware of the protective shell is protected by code-signing, allowing its operation only after its code integrity has been thoroughly checked. This stringent protection makes it significantly difficult for attackers to compromise the signing and forwarding process. Further details on the effectiveness of this code-signing approach will be elaborated in the specific protective shell example of the Jeep Cherokee security incident in section 5.5.3. The combination of these two mitigation techniques ensures that the monitoring systems receive a "clean feed" of the centrifuges' operational data, which is not susceptible to manipulation by attackers. However, it is essential to recognise that if the monitoring systems themselves are infected, the protective shell's influence on ensuring the accuracy of the displayed data would be limited.

Alerting and Logging Similar to previous protective shells, the specific protective shell designed for the uranium-enriching centrifuges incorporates essential alerting and logging capabilities. These features play a critical role in maintaining the security and integrity of the CPS by ensuring prompt responses to potential threats and enabling post-incident analysis. Whenever the protective shell detects abnormal behaviour or has to intervene with the CPS intended actions, the personnel must be alerted through their monitoring systems and extensive logs have to be written. By receiving real-time alerts, personnel can take immediate actions to investigate and respond to potential security breaches or impending failures. Additionally, the extensive logs are invaluable during post-incident analysis, as they can provide insights into the goals of the cyber-attack, or in case of predictive maintenance, what components are prone to failures and need replacement frequently.

5.3.4 Protective Shell Evaluation

This section presents a critical assessment whether the malicious destruction of uranium-enriching centrifuges during the Stuxnet cyber-attack could have been prevented through the implementation of a specific protective shell, equipped with previously presented mitigation mechanisms. The primary objective of the protective shell is to safeguard the centrifuges controlled by PLCs from destruction due to malicious instructions sent to them. Consequently, the protective shell must address the identified vulnerabilities in the original design architecture uncovered during the incident analysis.
The evaluation aims to determine the effectiveness of the protective shell's capabilities in mitigating the identified vulnerabilities through various monitoring and validation processes, thereby reducing residual risk. Additionally, it investigates whether the implementation of the protective shell introduced any new vulnerabilities.

Remediation of Old Attack Vectors The original centrifuges controlled by Siemens Simatic-S7 PLCs lacked crucial range validations, allowing Stuxnet to manipulate the centrifuges rotations outside their safe operation frequency. To counter this, the first important mitigation mechanism incorporated in the specific protective shell provided the CPS with the safety-critical range validations. Instructions send to the centrifuges that violate these range validation are intercepted and discarded by the protective shell, mitigating potentially dangerous outcomes.
Furthermore, more sophisticated anomaly detection through a behavioural model provides the protective shell with additional capabilities for detecting rather concealed malicious activity and enables predictive maintenance. Consequently, deviations from the normal uranium enrichment process, whether caused by unauthorised activities or failing components, can be promptly and efficiently detected, enhancing the overall system's reliability and trustworthiness. This approach reduces failure modes arising from unpredicted maintenance or malicious actions.
Moreover, the protective shell introduces improvements in the protection of operational data forwarded to external monitoring systems. This enhancement ensures more reliable and trustworthy monitoring for personnel overseeing the centrifuges controlled by PLCs and mitigates the risk of falsified operation data being used for distraction.
Lastly, through the incorporation of alerting and logging capabilities, personnel can rapidly respond to detected intrusions, limiting potential damages at an early stage. Additionally, the logs aid in post-incident analysis, enabling the discovery of primary attack goals, exploited vulnerabilities, and informing strategies to reinforce security solutions. This remediates the failure mode observed during the Stuxnet attack, where the malware operated undetected for several months.

Identification of New Vulnerabilities Similar to the specific protective shell for the Therac-25, the mitigation techniques employed here impose more stringent operational parameters compared to the original design. Consequently, identifying new attack vectors or failure modes becomes challenging. One potential concern is that the protective shell might falsely interrupt the current uranium enrichment process, if an anomaly is detected, causing the current gas batch to be unusable. However, it is essential to prioritise the protection of the centrifuges than trying to save the gas batch and risking a more permanent set-back of the nuclear program.

Conclusion The evaluated protective shell incorporates multiple mitigation mechanisms, significantly enhancing the system's trustworthiness. It effectively addresses the vulnerabilities found in the PLCs controlling the centrifuges. However, since Stuxnet affected many more systems along its way through the nuclear facility network, this protective shell only adds safeguards to the centrifuges and the Siemens controllers.
Considering the evaluation, it is highly plausible that the Stuxnet's primary goal of destroying the centrifuges could have been prevented with the implementation of this suggested protective shell, thereby reducing the risk of compromise and potential property damage to critical uranium-enriching centrifuges.
The thorough evaluation of this protective shell's features and their impact on avoiding failure modes underscores the significance of such security measures in CPS, particularly in critical domains like military and research.

5.4 Toyota "Unintended Acceleration" ETCS

5.4.1 Introduction

Starting in 2002, there was an increased number of reports regarding unintended acceleration (UA) in newer Toyota vehicles, resulting in instances of uncontrollable vehicles and subsequent crashes. Initially, Toyota attributed these incidents to their floor mats, suggesting that they were causing UA by holding down the acceleration pedal. Consequently, Toyota initiated a recall of floor mats in an attempt to mitigate the occurrence of UA accidents.
However, there were cases where floor mats could not have caused the accidents, either because they were stored in the trunk or were not used at all.
The second component implicated by Toyota was the acceleration pedal itself. The so-called "sticky gas pedals" were believed to cause UA due to complete failure of the spring responsible for returning the throttle, or insufficient strength to do so. Despite an expanded recall encompassing these "sticky gas pedals", it failed to significantly reduce the number of UA accidents.
Fast-forward to 2010, CBS News reported a total of 6,200 complaints, 57 injuries, and 89 deaths, all of which were correlated with Toyota UA incidents. As a result, a congressional investigation was launched against Toyota, with a team at NASA assigned to examine the technical aspects of UA, including the electronic throttle control system (ETCS). ETCS operates by electronically decoupling the direct mechanical connection between the accelerator pedal and the engine, with the pedal position directly influencing engine power. Consequently, ETCS governs air and fuel flow within the system, as well as the ignition spark to regulate engine power. Undeniably, ETCS is a safety-critical system, as its failure can lead to fatalities, injuries, and significant property damage.
During subsequent court trials involving Toyota, technical experts testified that the malfunctioning of the ETCS was likely the cause of many UA accidents. They discovered software glitches and hardware failures, leading the jury to side with the prosecution against Toyota. Furthermore, these trials revealed that Toyota had not provided NASA with all the necessary information regarding ETCS during the congressional investigation and had, on multiple occasions, misrepresented the safety mechanisms implemented.
Engineering principles such as fail-safe design, redundancy, and system testing were disregarded or ignored in the design of the ETCS. The Toyota UA trials served as a clear demonstration that corporations can be held accountable for negligently disregarding safety standards and developing unreasonably dangerous products for their intended use. Moreover, the presence of safety regulations, akin to those governing the aviation sector under the Federal Aviation Administration, is not a prerequisite for prosecuting corporations in sectors lacking such regulations. In sectors like the automotive industry, where no central authority enforces specific regulations, it is sufficient to establish corporate liability by demonstrating the defect's existence in the intended use of the product and establishing the link between this defect and the security incident or safety accident.

5.4.2 Vulnerabilities within CPS

Vulnerabilities within Toyota's ETCS were found by NASA commissioned by US Department of Transportation [25] and during the Bookout versus Toyota & Schwarz versus Toyota trials [26].

Single Point of Failure In Toyota's ETCS design, there are two independent sensors, VPA1 and VPA2, which serve as inputs for the accelerator pedal position, as depicted in Figure 5.3. This design aims to provide redundancy, wherein if one sensor fails, the monitor ASIC can

identify any discrepancies in sensor readings and take appropriate action. However, both signals are routed to the same analogue/digital converter, creating a single point of failure. If the converter malfunctions, incorrect calculations for both sensors are transmitted to the main CPU, rendering the redundant schema ineffective. The main CPU lacks the capability to validate the input from the monitor ASIC since the sensor readings have already been calculated, which may result in falsified results. Consequently, a malfunction in the converter could lead to an electronically stuck accelerator pedal.

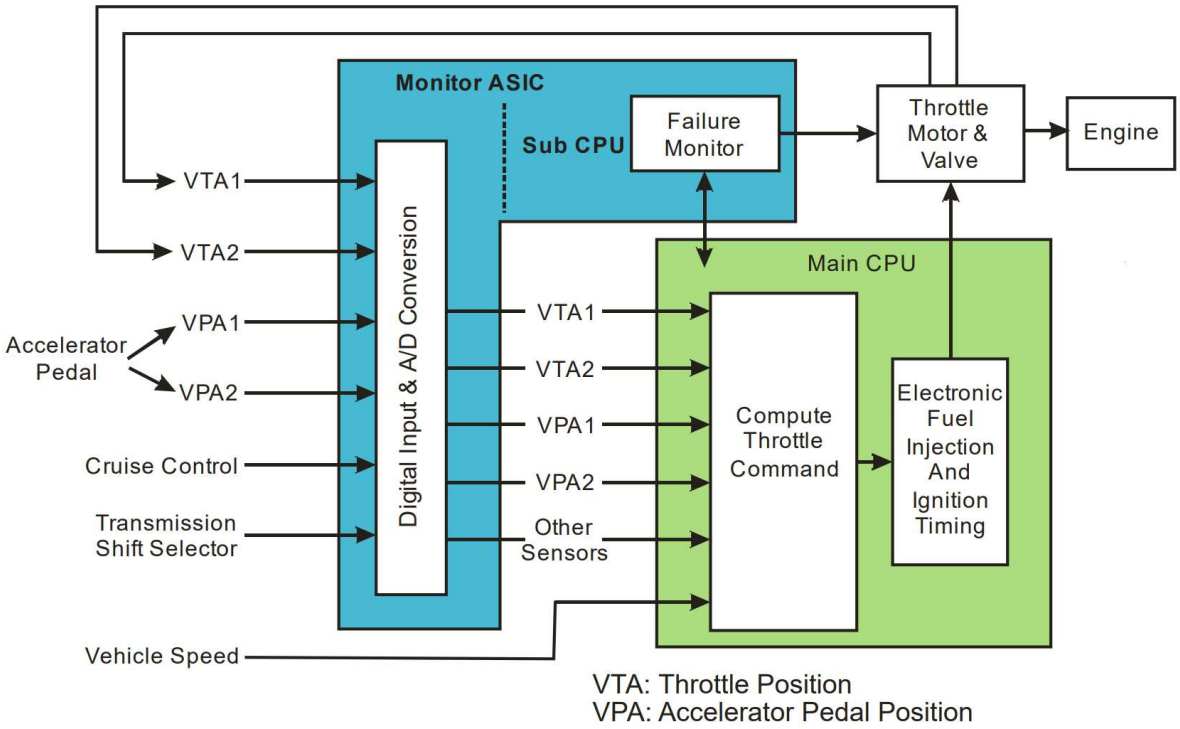


Figure 5.3: Toyota UA - Single Point of Failure [25]

No Brake Override In the Toyota models affected by unintended acceleration, the braking system was power-assisted, allowing the driver to apply only a fraction of the force required on the brake pedal, with the remaining force provided by a vacuum pump to achieve the necessary braking power. However, when the throttle was engaged, the brakes were no longer power-assisted, resulting in a reduction of the force applied to the brakes by approximately 8 to 10 times. NASA calculations revealed that in order to compensate for full-throttle acceleration, the driver would need to exert a force of 80 kg on the brake pedal [25].
Furthermore, when the throttle was engaged, the brakes not only lacked power assistance but also did not override the throttle input. This means that there was no mutual exclusion implemented in the ETCS to prevent simultaneous acceleration and braking. In a robust safety architecture, the brake signal should take priority over the accelerator signal, ensuring fail-safe operation in the event of an electronically stuck accelerator pedal.

No Error Correcting Code for RAM Toyota informed NASA that they had implemented error detection and correction (EDAC) for the static RAM used in the ETCS. This involved incorporating additional bits in memory to guard against bit errors by utilising error-correcting code algorithms, which can detect and sometimes even correct such errors.
However, it was later discovered that Toyota had not actually implemented EDAC for the SRAM utilised in the ETCS. Given that the ETCS is a safety-critical system, a single bit flip could

potentially lead to an accident. Bit flips can occur due to software defects or as a result of a phenomenon known as a single event effect. Single event effects refer to the continuous radiation of highly energetic particles in the surroundings, which can impact sensitive nodes and cause them to change state (i.e., experience a bit flip).
Both causes, software defects and single event effects, must be effectively mitigated in safety-critical systems to prevent run-time errors that could result in unpredictable outcomes.

Stack Buffer Overflow and Software Defects In the Bookout trial, experts concluded that the ETCS code contained software defects that led to stack buffer overflow [26]. Stack buffer overflow is a software vulnerability that occurs when a program writes more data into a fixed-size buffer on the stack than it can hold. As a result, it can overwrite data at addresses used by other functions, leading to unexpected behaviour.
One of the identified software defects was the use of unprotected recursion, resulting in memory corruption. As mentioned earlier, this memory corruption was not protected by EDAC and could lead to the suspension of critical ETCS tasks. Furthermore, some of these task suspensions were not detected by fail-safe mechanisms that should have restarted the entire system. These gaps in fail-safe behaviour rendered the ETCS software less robust, as failures in either hardware or software components could compromise the system's overall safety.
Undoubtedly, there are likely numerous other software defects present in the ETCS. In the Bookout trial, experts discovered approximately 80,000 violations of MISRA C rules within 296,100 lines of code [26]. MISRA C is a coding standard for the C programming language that provides guidelines and rules to ensure the safety, reliability, and portability of embedded software. It aims to minimise potential vulnerabilities and improve code quality by enforcing strict rules regarding language usage, syntax, and coding practices. As a general guideline, it is often estimated that for every 30 MISRA C rule violations, the software likely contains one major bug.

5.4.3 Specific Protective Shell Mitigation Mechanisms

The specific protective shell developed for the Toyota electronic throttle control system (ETCS) incorporates several core mitigation mechanisms to address the vulnerabilities that led to the accidents previously analysed. The purpose of this protective shell is to safeguard the CPS electronically controlling the throttle from failures. In the following section, those key mitigation mechanisms are presented, which build the main characteristics of this protective shell.

Reevaluation of Sensor Readings One critical vulnerability in the original ETCS design was attributed to a single-point-of-failure in the analogue/digital converter, leading to potential conversion errors and an electronically stuck acceleration pedal. To address this core vulnerability, the protective shell incorporates a fundamental mitigation mechanism.
As depicted in the protective shell's architecture 4.3, it actively monitors all sensor readings, including the two accelerator pedal positions (VPA) and the application signals sent to actuators, such as the throttle position (VTA) transmitted to the engine. Given the high correlation between the accelerator pedal and throttle positions, the protective shell can effectively detect anomalies by comparing the movements of these parameters. Typically, pressing the accelerator pedal down results in an increased throttle, and vice versa. However, if the accelerator pedal is moved up while the throttle position remains unchanged, the protective shell identifies an anomaly and intervenes by adjusting the signals sent to the engine to reduce power accordingly. Additionally, redundancy is introduced through the utilisation of

two sets of sensors for VPA and VTA, which are continuously validated for any deviations between them. This redundancy ensures that the protective shell is not reliant on a single sensor, mitigating potential single-point-of-failures.
Moreover, there are several other correlations between the accelerator pedal sensor and other variables, such as engine torque and speed. These correlations enable cross-validations that can be incorporated into the predefined desired behaviour by domain experts, either through specifications and rule-sets or as part of the trained behaviour baseline, which will be discussed in more detail below.
To enhance robustness, the protective shell may include a redundant A/D converter to continuously monitor the functionality of the primary converter within the ASIC chip as another mitigation approach to failures. While the additional computation overhead and value comparison introduce some latency compared to the more efficient cross-validation and plausibility checks, this redundant, fault-tolerant setup completely mitigates the single-point-of-failure vulnerability.

Brake Signal Priority An additional essential mechanism in preventing unintended acceleration is signal prioritisation or more specifically brake override. By implementing mutual exclusion for simultaneous acceleration and braking, with the brake signal taking priority, the protective shell can effectively prevent the vehicle from accelerating when the brake is applied, providing the driver with a means to intervene in hazardous situations.
It is essential to note that there is no practical scenario where the driver would need to accelerate and brake simultaneously. Thus, when both acceleration and brake signals are received simultaneously by the protective shell, an anomaly is detected, triggering corrective actions to prioritise the brake signal. Since accelerating and braking are exactly counteracting, a false signal could equalise or diminish the correct signal and its actuator actions. Moreover, the brake signal does normally lead the system into a safe state, in contrast to uncontrolled acceleration.
As part of the sensor readings reevaluation within the model specified by domain experts, the protective shell incorporates the understanding of the inverted nature and direct correlation between the accelerator and brake signals. By recognising this relationship, the protective shell can promptly discard any acceleration request to the ETCS when brakes are pressed, further enhancing the safety and responsiveness of the system.

Behaviour Model Training In line with other protective shells, Toyota's ETCS can significantly benefit from implementing a behaviour baseline that extends the static models specified by domain experts and is trained using real-world operation data from the vehicles. While the concept of the behaviour baseline has been explained in detail previously, it is worth highlighting the specific advantages it brings to the implementation of Toyota's ETCS. The behaviour baseline plays a crucial role in the reevaluation of operation data, facilitating cross-validation against known correlations, and conducting thorough plausibility checks. By analysing historical data and comparing it to established behaviour patterns, the behaviour baseline helps ensure that the system operates as expected and identifies any deviations that may indicate potential anomalies or hazards.
Furthermore, the behaviour baseline provides an effective means for detecting emergent, negative properties that may not have been anticipated during the system's development. A prime example is the unpredicted suspension of critical ETCS tasks due to software defects or memory corruption. In such cases, where certain tasks lack monitoring mechanisms (e.g., a watchdog timer) to ensure their proper operation, the behaviour baseline can step in and acquire intrinsic knowledge about the expected behaviour of these tasks through analysis of application events recorded in log files. With this knowledge, the protective shell becomes equipped to detect anomalies caused by software defects, memory corruption, or

single event effects. If any such anomalies are detected, the protective shell can respond promptly and appropriately by taking corrective actions. For instance, in the event of a critical ETCS task failure, the protective shell first detects it and may attempt to restart the failed task or even initiate a system-wide restart, mirroring the actions of a watchdog mechanism. By leveraging anomaly detection through a trained behaviour model, Toyota's ETCS can enhance its ability to anticipate and address unexpected issues, ultimately enhancing the system's trustworthiness while ensuring its continued reliability during real-world operation.

Time-Sequential Analysis and Retrospective Data Evaluation Another crucial mitigation mechanism employed in the protective shell is time-sequential data analysis, similar to the approach implemented in the Boeing 737 Max MCAS protective shell. By considering previously received data streams, the protective shell can assess whether the ETCS is experiencing or heading towards a failure mode. For example, when examining data related to an electronically stuck acceleration pedal, a single data point may not immediately reveal an anomaly. However, when viewed in the context of previous values, non-fluctuating readings would indicate that something is wrong, as human drivers cannot consistently maintain the acceleration pedal at the same position for an extended period, except for full throttle or no throttle. Time-sequential analysis empowers the protective shell to identify deviations and anomalies by tracking data trends and variations.
Furthermore, time-sequential analysis enables the protective shell to counteract unpredictable algorithm outcomes arising from software defects, memory corruption, or similar issues. Algorithms are inherently deterministic, meaning they should produce the same output for the same input. Consequently, application signals should correlate strictly with sensor readings. By analysing historical data vectors combining sensors, actuators, and application signals and leveraging the trained behaviour model, the protective shell can project these data sets into the near future and determine individual value expectations and acceptable thresholds (using data trends and variations as references). For instance, if a vehicle is travelling at 50 km/h and the protective shell has a sampling frequency of 100 Hz (acquiring 100 data points per second or 1 data point every 10 ms), the protective shell can calculate the expected speed value for the next iteration.
Suppose the vehicle is exhibiting an accelerating trend, and the protective shell calculates an expected speed value of 50.2 km/h for the next data point. However, considering the possibility of the driver initiating emergency braking or accelerating more rapidly, the protective shell establishes possible thresholds based on the expected value, such as 48 km/h as the lower bound and 50.5 km/h as the upper bound. Consequently, when the next data vectors are reported to the protective shell, it can efficiently and rapidly assess whether the values can be trusted or if they indicate anomalies. Values that are reported as impossible or highly unlikely based on the established thresholds will prompt the protective shell to take appropriate actions to address potential issues and ensure the system's continued safe operation.

Alerting and Logging Alerting and logging capabilities are also required for this protective shell but deviate a bit from the others CPS previously introduced. In the context of vehicles, the protective shell must be equipped with alerting and logging capabilities to keep passengers and drivers informed about the system's status and any detected anomalies. As the majority of drivers are not experts in vehicle mechanics, the protective shell needs to provide clear and concise alerts displayed on the infotainment display. These alerts should not delve into excessive technical detail but rather deliver essential information, such as the necessary steps to be taken (e.g., visiting a service centre) or if the protective shell has already taken corrective actions.

Real-time information is crucial, as it empowers drivers to assess the situation and take appropriate protective measures on their own. By receiving timely alerts, drivers can quickly understand the potential risks and make informed decisions to ensure their safety and the safety of their passengers.
On the other hand, the operational logs generated by the protective shell must be very detailed. These logs serve as a valuable resource for experts who analyse them to identify various scenarios. Experts need to be able to distinguish between an isolated problem that affects only one vehicle, a broader issue affecting a fleet of similar vehicles, or instances where the protective shell may have triggered a false alarm.
By striking the right balance between clear and concise alerts for drivers and detailed logs for experts, the protective shell can enhance the overall safety and security of the vehicle. Drivers can respond promptly to potential risks, while experts can gain insights from the logs to address specific anomalies and improve the system's reliability.

5.4.4 Protective Shell Evaluation

This section presents a rigorous evaluation of whether the "Unintended Acceleration" accidents in Toyota vehicles could have been prevented through the implementation of a specific protective shell equipped with the previously described mitigation mechanisms. The primary objective of the protective shell is to safeguard Toyota's ETCS from faults that have been identified as contributing factors to safety accidents resulting in numerous injuries and fatalities. Therefore, the protective shell must effectively address the vulnerabilities in the original ETCS design that were uncovered during the accident analysis.
The evaluation seeks to assess the effectiveness of the protective shell's capabilities in mitigating the identified vulnerabilities through a series of monitoring and validation processes, ultimately reducing the residual risk associated with potential failures. Additionally, the evaluation investigates whether the implementation of the protective shell introduced any new vulnerabilities that could compromise the overall safety and reliability of the system.

Remediation of Old Failure Modes The protective shell is equipped with several effective mitigation mechanisms to address the failure mode of an electronically stuck acceleration pedal. If the A/D converter fails, causing the main CPU to erroneously perceive the acceleration pedal as being in a fixed position for an extended time, the protective shell would identify this anomaly through monitoring and validation processes.
Firstly, the reevaluation of the acceleration pedal signal in relation to the throttle level serves as a key indicator of abnormal behaviour, prompting the protective shell to intervene immediately. Secondly, cross-validation to several correlated data streams to the throttle position enables the protective shell to detect an unresponsive engine power control to the pedal, thus triggering safety fallback measures, such as safely bringing the vehicle to a stop. Thirdly, the time-sequential analysis and anomaly detection based on the behaviour model reinforce the protective shell's identification of an electronically stuck pedal condition. The presence of multiple methods to underline an anomaly ensures a sound and reliable assessment by the protective shell, thereby enhancing the overall trustworthiness of the system and providing increased explainability of its actions.

Another critical failure mode addressed by the protective shell is the lack of power assistance in the brakes during unintended acceleration, making it challenging for the driver to apply sufficient pressure on the brakes to counteract the acceleration force and slowing the vehicle down. The implementation of brake override in the protective shell guarantees that any throttle signal is overridden by brake signals, swiftly stopping ongoing acceleration and enabling the vacuum pump to provide the intended power assistance for the brakes.

The safety accidents resulting from this fault were not instantaneous, as drivers typically had enough time to react and take protective measures on their own. For instance, in the tragic crash on August 28, 2009, involving driver Mark Saylor, a highway patrol officer and vehicle inspector, and three passengers falling victim to an unintended acceleration accident while being on an emergency phone call with 911. This shows that the driver had sufficient time to react to the upcoming safety accident if he could initiate an emergency call while the vehicle accelerated. The protective shell provides the driver with the capabilities to initiate strong protective actions by enforcing brake override.

Furthermore, the protective shell seeks to prevent failure modes associated with software defects, memory corruption, and other similar issues. The unpredictability of algorithm outcomes in certain operation conditions, caused by single event effects, software bugs (indicated by numerous MISRA C rule violations in the ETCS' code base), and memory corruption, has the potential to lead to faults during run-time, increasing the risk of safety accidents. While the protective shell does not directly improve the underlying architecture of the run-time core, such as implementing much-needed Error Detection and Correction (EDAC) for the SRAM, it deploys mitigation mechanisms to detect and rectify safety-critical tasks that exhibit unpredictable behaviour before they propagate into system-wide negative effects.
The behaviour model's anomaly detection plays a central role in identifying early signs of system drift towards failure, analysing the origins of negative properties, and taking appropriate corrective actions, such as isolating task signals to actuators and restarting them to address run-time errors. By actively monitoring and addressing such issues, the protective shell contributes significantly to maintaining the overall safety and reliability of the ETCS.

Identification of New Vulnerabilities While the protective shell offers substantial benefits in mitigating known failure modes, it is essential to consider potential new vulnerabilities that may be introduced by its implementation to ensure a thorough evaluation of its effectiveness for the CPS.
One new failure mode introduced with the implementation is caused by the protective shell falsely detecting anomalies, such as unintended acceleration, leading to the gradual reduction of engine power and bringing the vehicle to a stop. In such a scenario, the effectiveness of the protective shell depends on the timely response of the driver and passengers to the alerts displayed on their infotainment system's display. If they fail to react promptly, stopping the vehicle on an active lane instead of the hard shoulder could significantly increase the risk of rear-end collisions and potential hazards to other road users.
Furthermore, the prioritisation of brake signals by the protective shell may render the vehicle non-driveable if incorrect brake signals are received. This situation can pose serious risks to the occupants, particularly in critical driving conditions where the vehicle's responsiveness is crucial for safety.
Additionally, the automatic restart of individual tasks or the entire system during operation is another aspect that requires careful consideration. While restarting can be a valuable recovery mechanism for resolving certain failures, it must be approached with caution. Automatic restarts should only be employed if the alternative presents a greater risk, and the protective shell must be capable of distinguishing between transient errors and persistent faults. For instance, if a non-critical task fails, it might be safer to display a warning to the driver to restart the vehicle safely, rather than automatically restarting specific tasks. Additionally, if the error persists even after a restart, the vehicle should ideally be outside road traffic, in a safe location where the driver can seek expert assistance.
To ensure the overall effectiveness of the protective shell and maintain a high level of safety, these potential new failure modes should be thoroughly assessed, and suitable safeguards

should be implemented to mitigate their impact. Striking a balance between proactive intervention and ensuring human control and decision-making is critical in achieving a reliable and robust protective shell for this specific CPS.

Conclusion The evaluated protective shell incorporates multiple mitigation mechanisms, significantly enhancing the system's trustworthiness by effectively addressing most of the vulnerabilities found in Toyota's ETCS. However, some vulnerabilities in the run-time CPS cannot be entirely eliminated by the protective shell alone, but their risks they are exposing for failures can be substantially reduced. For instance, the protective shell cannot provide the underlying CPS run-time system with error detection and correction techniques for their SRAM.
Notably, the protective shell introduces a few new failure modes; nevertheless, their risks remain within acceptable ranges and are considerably lower than the failure modes that have been successfully remediated. Overall, the implementation of the protective shell reduces the residual risk to an acceptable level.
Considering the evaluation, it is highly plausible that many safety accidents associated with ETCS failures could have been prevented with the suggested implementation of this protective shell, thus significantly reducing the risk of injuries and fatalities in Toyota vehicles caused by unintended acceleration. The thorough assessment of the protective shell's features and their impact on avoiding failure modes underscores the importance of such safety measures in CPS, particularly in critical domains like the automotive industry.

5.5 Jeep Cherokee Hack

5.5.1 Introduction

The Jeep Cherokee Hack was a well-known security incident in the automotive industry and caused the first physical recall of a mass-produced product because of a software security issue.
In 2015, two security researchers demonstrated that they were able to remotely take control of a Jeep Cherokee through its Uconnect infotainment system, which had besides others a cellular connection to the internet. The researchers were able to access the vehicle's critical systems, including the brakes and steering. The ethical hack was conducted by Charlie Miller and Chris Valasek, without endangering any persons or causing property damage. The researchers informed Chrysler of the vulnerability, and the company issued a recall for 1.4 million vehicles to apply security patches.
The incident raised concerns about the security of connected vehicles and the potential for cyberattacks. It highlighted the need for automotive manufacturers to pay closer attention to the security of their products and ensure that appropriate security measures are in place. While adding more and more functionalities and increasing usability of their products, the attack vector increased drastically and the sheer amount of code made software vulnerabilities almost certain. Therefore, a safety- and security-aware development process is required while developing safety- and security-critical systems and leverage the principle of safety and security by design. Otherwise, safety and security will fall behind while the developers try to meet time-to-market requirements with high management expectations and low overall budget.

5.5.2 Vulnerabilities within CPS

The following vulnerabilities are summarised and carefully selected from Charlie Miller's and Chris Valasek's extensive report, "Remote Exploitation of an Unaltered Passenger Vehicle" [27].
In a modern car, the attack surface is significantly large. In addition to wired connections such as USB ports, there are numerous wireless connections. Nowadays, the multimedia system is often equipped with Bluetooth and W-iFi capabilities to facilitate easy connectivity with mobile devices. Moreover, some cars have cellular connections for Over-the-Air firmware updates. Lastly, certain internal components, like the tire-pressure monitoring system (TPMS), are also connected wirelessly. Consequently, an attacker has a wide range of potential intrusion points.
Upon successful infiltration, the ultimate objective is to transmit arbitrary controller area network (CAN) messages. These messages are sent and received on the CAN bus by vehicle components to facilitate communication among themselves. While some messages pertain to less safety-critical components like the radio or navigation system, there are also highly safety-critical components involved, such as ABS, the engine, and the self-parking assistant.

Weak WPA2 Password Generator As mentioned previously, one attack vector involves the built-in Wi-Fi Hotspot. Uconnect utilised the WPA2 security protocol standard, which is still considered secure today. In order to establish a connection to the Wi-Fi Hotspot, the user had to enter a seemingly random password displayed on the infotainment system's screen. However, this purportedly "random" password was not truly random.
To begin with, it is important to understand that computer algorithms do not possess true randomness in and of themselves. Algorithms are deterministic, meaning that for the same input, an algorithm will always produce the same output. Generating a genuinely random

number is a non-trivial task for a computer. Various methods exist to simulate a form of non-determinism. One popular approach involves initially seeding the algorithm. A seed is an additional input that should incorporate pseudo-randomness, which the algorithm cannot generate on its own. By inputting a seed, the algorithm can exhibit randomness equivalent to that of the original seed.
Returning to the issue of weak WPA2 password generation in the Uconnect system, one may speculate that the chosen seed for the algorithm was flawed. Uconnect seeded the generation process based on a timestamp in seconds from when the vehicle, specifically the infotainment system, was first booted. Consequently, an attacker could significantly reduce the potential password space to approximately 15 million passwords if they could determine the production year. Furthermore, by imposing the condition that the car was initially turned on during the day, the attacker could further decrease the number of possible passwords to 7 million. Even as early as 2015, it took merely an hour to brute force 7 million WPA2 passwords, with an expected time of approximately 30 minutes to find the correct password.
However, the password generator was discovered to be far more insecure. The infotainment system required a cellular connection to synchronise the current time with a server during its initial boot. Unfortunately, many vehicles were started within the factory for testing purposes, where a cellular connection was either unavailable or unstable. Since time is crucial for numerous functionalities, the developers decided to set the time to a default value (01.01.2013) if no cellular connection could be established during the first boot. Even if the user manually adjusted the time later on, it would not alter the static WPA2 password once it was set. Consequently, most vehicles ended up generating one of a few WPA2 passwords, with the first few seconds of 2013 as the seed.

Insecure Configuration When connected to the Uconnect local network, an NMAP scan can be initiated to search for hosts and open ports. The car's Wi-Fi interface reveals multiple open ports, one of which belongs to the D-Bus service. D-Bus facilitates interprocess communication among different internal applications. Therefore, it is essential to have a firewall in place to block external communication.
Participating in D-Bus communication usually requires authentication, limiting access to trusted entities. However, the D-Bus was configured to allow anonymous logins and unrestricted participation in the communication.
Next, the attacker needs to acquire command line execution capabilities on the car's head unit, possibly through the running D-Bus service. Security analysts Charlie Miller and Chris Valasek discovered several opportunities for command line injection within the methods provided by the D-Bus services. By crafting a parameter that escapes the current context with special characters, the system would interpret the remaining parameter as bash commands.
However, upon analysing the potential D-Bus service methods, they found a standard method called "execute" that simply executes the given parameter on the command line. As a result, no command line injection was necessary.
Lastly, to infiltrate the system further and establish persistence, the typical next step would involve exploiting privilege escalation vulnerabilities to elevate the permissions of the currently running shell command. However, the D-Bus service was executed as root, granting full system permissions.

Later, it was found that not only did the Wi-Fi interface have several ports open, but also the cellular interface. Therefore, the entire attack pattern was not only possible from a close distance via the Wi-Fi connection but enlarged the attack vector to a global scale by assigning the car host an IP address.

No Firmware Signing As previously explained, the ultimate objective is to transmit arbitrary CAN messages on the interior high-speed CAN bus in order to control devices that are listening on this bus. Car manufacturers promote the security of their vehicles by highlighting the built-in air gap between the infotainment system chip, which has extensive connectivity, and their safeguarded safety-critical systems.
This secure design principle is absolutely crucial in mixed critical systems. However, what they fail to mention is that the infotainment chip is not completely isolated from the CAN bus to the extent one might assume when hearing the term "air-gapped." While it is true that this chip cannot directly send CAN messages, certain functionalities of the infotainment system require CAN connectivity. Consequently, there is another chip on the board that is connected to both the CAN bus and the infotainment chip. When the infotainment system requires a functionality covered by specific CAN messages, it communicates with the second chip through their Serial Peripheral Interface (SPI) connection, and the second chip then sends out the required CAN messages on its behalf.
There is no inherent capability to send arbitrary CAN messages from the infotainment chip. However, since users can update the firmware by connecting a USB stick to the infotainment system's USB hub, there must be an update mechanism initiated from the first chip to the second chip with CAN bus connectivity. One could imagine manipulating the firmware of the second chip in such a way that a backdoor is implemented, enabling the infotainment chip to transmit arbitrary CAN messages to the CAN bus instead of just predefined ones that the second chip would forward. Typically, manipulated firmware is non-functional because during startup, its integrity and authenticity are verified by validating the included signature. However, in cars using the Uconnect system, no code signing measures were in place to mitigate such an attack.

Ultimately, security analysts Charlie Miller and Chris Valasek successfully managed to transmit arbitrary CAN messages and gain control over significant portions of the car's digital components. They were able not only to fully control the multimedia system, air conditioning, and GPS tracking but also to access safety-critical systems such as the brakes, speedometer, steering, lock mechanism, and even shut down the engine.

The insecure software within the Uconnect system, developed by Harman International, can be considered a supply chain vulnerability that impacted a wide range of Fiat Chrysler Automobiles brands, such as Jeep, Dodge, and RAM. This serves as a fitting illustration of the disregard for Zero Trust Architecture across multiple interfaces, where communication lacked both security and authentication measures. It is crucial to isolate critical systems from less secure and easily accessible systems, such as infotainment, in order to minimise the potential harm resulting from a single security breach.

5.5.3 Specific Protective Shell Mitigation Mechanisms

The specific protective shell developed for the Uconnect infotainment system incorporates several core mitigation mechanisms to address the vulnerabilities that led to the security incident previously analysed. The purpose of this protective shell is to safeguard the CPS from attacks, that controls some important features in the car and has indirect access to the internal CAN bus communicating to safety- and security-critical components. In the following section, those key mitigation mechanisms are presented, which build the main characteristics of this protective shell.

Brute-force Attack Detection One of the important mitigation mechanisms employed to address cyber-attacks on the Uconnect system is the implementation of brute-force attack

detection and prevention. Brute-force attacks involve repeated attempts to guess a password or key in order to gain unauthorised access. Although this mechanism may be considered relatively insignificant in the context of the cyber-attack analysed previously, it plays a crucial role in a broader context for the Uconnect system security and its protection.
A key challenge for the protective shell is that it lacks direct control over the generation process of the WPA2 password and cannot modify design decisions related to using the system time as a seed and its pseudo-randomness. Consequently, the protective shell relies on sophisticated data analysis of packets traversing through it to identify and restrain undesired Wi-Fi access attempts by potential attackers. To accomplish this, the protective shell adopts mechanisms commonly employed in network-based Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS). Two primary techniques are utilised: rate limiting and address blocking.
Rate limiting is employed to restrict the number of login attempts processed within a defined time-frame. By artificially slowing down the processing rate of login attempts, brute-force attacks are rendered impractical, as the sheer volume of possible combinations makes it infeasible to guess the correct password within a reasonable timeframe.
Address blocking is another crucial mechanism utilised to counter brute-force attacks. It involves temporarily banning addresses that exhibit suspicious behaviour, such as multiple failed login attempts. By blocking access from these suspicious addresses, the protective shell mitigates the risk of unauthorised access attempts.
To specifically address brute-force attacks targeting the Uconnect system, the protective shell forwards only a limited number of authentication requests to the Wi-Fi module per given time-frame, discarding any excess requests. By doing so, the protective shell effectively attackers from trying different passwords rapidly. Furthermore, the protective shell has the capability to detect suspicious behaviour from specific addresses and block further communication, particularly in cases where attackers do not employ MAC address spoofing. By identifying and isolating these suspicious addresses, the protective shell enhances the overall resilience of the Uconnect system against brute-force attacks.

Emulated Firewall Following the publication of the report by Miller and Valasek [27], Fiat Chrysler released a security update for the Uconnect system to rendering the exact attack pattern from Miller and Valasek ineffective. This update involved a key mitigation mechanism, the restriction (status filtered) of ports on the Uconnect interfaces for both Wi-Fi and cellular connections to external devices. The protective shell can adapt this technique by effectively emulating this firewall behaviour, thereby mitigating the insecure configuration that existed in the original Uconnect design.
The protective shell acts as a filtering barrier, intercepting and discarding network communication packets from external devices that are intended to reach internal components of the Uconnect system. This strategic filtering prevents direct communication with internal components from external devices, which was not an intended use case in the original Uconnect design. By imposing this restriction, the protective shell significantly reduces the attack vector available to potential malicious users.
In the original Uconnect design, certain open ports allowed external access to core system components, including the D-Bus service. However, with the implementation of the artificial emulated firewall in the protective shell, external devices are prevented from reaching these core system components through open ports. This restriction effectively limits an attacker's ability to infiltrate and propagate through the Uconnect system, thereby enhancing its overall security posture.
By discarding network communication packets destined for internal components from external devices, the protective shell ensures that communication with critical system elements is tightly controlled and restricted following the zero-trust architecture principle. This proac-

tive approach is a crucial step towards safeguarding the Uconnect system from unauthorised access and cyber-attacks.

Behaviour Model Training Another critical mitigation mechanisms involves the training of a behaviour model, utilised for advanced anomaly detection, similar to the other specific protective shell discussed previously. One might state that in this case the implementation of strict firewall rules described above would completely remedy known attack vectors (like the incident analysed) and additional mitigation mechanisms are not necessary. That is of course not the case, since it is essential for the protective shell to follow a multi-layered defence-in-depth approach to protect against unknown attack patterns and eliminate vulnerabilities effectively. Assuming an attacker gains the capability to send D-Bus or CAN-Bus messages through an alternative attack path other than the Wi-Fi and cellular interface; the protective shell must counteract such malicious activity to prevent security-incidents.
A promising mechanism for detecting malicious activity in the Uconnect system's D-Bus and CAN-Bus communication is through anomaly detection based on a precalculated behaviour baseline derived from operation data and log files. As the communication in the Uconnect system follows relatively static patterns with similar payloads (e.g., the infotainment chip sending a D-Bus message to regulate air conditioning), a meaningful behaviour model can be established.
A primary indicator of an anomaly is the absence of built-in authentication functionality in packets sent to the D-Bus, if all standard components communicating used it. While it is possible for packets to be sent anonymously in the original Uconnect design, the lack of authentication raises immediate suspicion by the protective shell, leading to their discardation. However, it should be noted that there is no specific information provided regarding whether the components in the original design used authentication, rendering this technique useless if they would also anonymously.
Nevertheless, through deviation from the established behaviour model, packets sent by attackers can be quickly identified if they don't follow given operation trends during exploitation. For instance, if the payload of packets substantially differs from the normal payload associated with a specific message type (as is the case during command line injection attacks), the protective shell promptly detects this abnormal behaviour and prevents forwarding the packet to its intended receiver, safeguarding it against undesired effects.
Another deviation noticeable by anomaly detection could be the sudden use of message types (service calls) that are typically not utilised during normal operation. For instance, the execute service message should either never or very rarely be used during normal operation, as it is primarily intended as a debug tool for engineers to test and control the underlying system. Normally, all D-Bus functionalities are baked into service methods to be called, exposing and limiting the needed functionality for components. If the protective shell detects frequent use of this powerful execution tool during regular operation, it indicates suspicious behaviour that necessitates further investigation.
Moreover, anomaly detection can leverage correlations between messages in the Uconnect system to support security efforts. For instance, consider the process of unlocking the car with a remote. First, a radio package is sent to an internal receiver, authenticated and checked for replay-attacks. Upon successful validation, a CAN-Bus message is triggered to unlock the doors. If the protective shell detects a plain CAN-Bus message unlocking the doors without the preceding interaction with the radio receiver and validation processes, it indicates an abnormal behaviour that may require blocking.
Last but not least, this advanced anomaly detection capability might help to isolate attackers who have successfully infiltrated one component, preventing them from spreading to other components. Given the limited options for triggering messages on the CAN-Bus on behalf of the Uconnect system, the protective shell can detect unallowed CAN-Bus messages orig-

inating from the Uconnect system and block them from reaching other components. This isolation effectively contains the attacker's influence, when the CAN-Bus service got infiltrated, and limits the potential damage they can cause within the system.

Code Integrity Code and firmware signing is a critical mitigation mechanism that plays a significant role in safeguarding CPS during cyber-attacks. Code signing is a process that ensures execution is limited to trusted code only. In the context of the protective shell, this mechanism involves verifying the integrity and authenticity of its own code and that of the underlying CPS it encompasses during initial startup, and whenever there is a change to the executed code base.
To achieve this, asymmetric cryptographic operations are used, where a trusted party, such as the CPS manufacturer, signs the current code version of the protective shell with its certificate. This signing process prevents any undetected manipulations to the code and provides assurance that the running version of the protective shell was released by a trusted and legitimate source, the CPS manufacturer. For the protective shell to validate its own code base, it requires knowledge of the trusted manufacturer's public key necessary for verifying the code's signature.
It is important to note that the protective shell has no control over adding or adjusting code-signing practices for the underlying CPS application software and components firmware in the design architecture. As a result, this mitigation mechanism must be applicable to systems that run unsigned code, such as the Uconnect system. To address this, the protective shell maintains a trusted hash of the underlying CPS code base and recalculates and compares it during startup. By doing so, the protective shell ensures that the CPS operates only with trusted, unmodified, and integrity-proofed code. In cases where the code validation checks are unsuccessful, the protective shell intervenes and blocks the startup process, rendering the CPS non-functional. This approach not only enhances the security architecture of the CPS but also provides protection against safety accidents caused by corrupted firmware.
However, the Jeep Cherokee security incident highlighted a specific vulnerability wherein the attacker exploited the ability to update the firmware of the second chip connected to the CAN-Bus. To counteract this potential threat, the protective shell not only checks the firmware integrity upon startup but also verifies firmware integrity before permitting any firmware updates. When a firmware update request is sent from the infotainment chip to the second chip, the protective shell validates whether it recognises the new firmware hash before forwarding the firmware update request and allowing the update. Therefore, an update to the underlying code base does always require an update to the protective shells stored hashes, resulting in a resigning of the protective shells code. This authenticated update process ensures that any changes to the underlying software/firmware require approval and validation, providing an additional layer of protection against unauthorised modifications.
As a result, the implementation of the CAN-Message backdoor in the firmware would not be possible because any modification to the firmware would cause the firmware hash to change, violating its integrity. Consequently, the protective shell would not trust this altered firmware version and would effectively block the update.

Alerting and Logging Similarly to the mitigation mechanisms presented in the other specific protective shells, the Uconnect system safeguarded by the protective shell also needs alerting and logging capabilities for likewise reasons. The protective shell must inform the driver when it detects anomalies and executes protective actions. Alerts serve to notify the driver of potential cyber-attacks, prompting them to seek professional analysis. Detailed operation logs aid in the forensic analysis of security incidents, allowing the discovery of

previously unknown vulnerabilities and the improvement of overall system reliability and trustworthiness through future updates.

5.5.4 Protective Shell Evaluation

This section presents a rigorous evaluation of whether the "Jeep Cherokee Hack" security incident could have been prevented through the implementation of a specific protective shell equipped with the previously described mitigation mechanisms. The primary objective of the protective shell is to safeguard the Uconnect system from successful exploitation of the identified and analysed vulnerabilities. Therefore, the protective shell must effectively address those vulnerabilities in the original Uconnect design.
The evaluation seeks to assess the effectiveness of the protective shell's capabilities in mitigating the identified vulnerabilities through a series of monitoring and validation processes, ultimately reducing the residual risk associated with potential incidents. Additionally, the evaluation investigates whether the implementation of the protective shell introduced any new vulnerabilities that could compromise the overall security and reliability of the system.

Remediation of Old Attack Vectors The protective shell significantly reduces the effectiveness of brute-force attack targeted at the Uconnect's Wi-Fi module, thereby lowering the risk of attackers gaining unauthorised wireless access. With the presented brute-force attack detection and prevention mechanisms, the protective shell successfully remediates the exploited failure mode and the initial intrusion point in the attack pattern.
In case an attacker attempts to brute-force the WPA2 password, they would need to maintain close proximity to the driving car for an extended period due to the limited authentication tries per minute. Moreover, a more sophisticated brute-force attack would be necessary, requiring the attacker to constantly spoof their MAC address to avoid automatic blocking by the protective shell. Such an attack is easily detected by the protective shell, triggering an alarm for the driver to initiate manual protective measures, such as disabling Wi-Fi, and alerting them of a potential targeted attack.

Another critical vulnerability is mitigated by the protective shell's emulated firewall capability. In the original Uconnect system, an insecure configuration allowed external devices connected via Wi-Fi or cellular network to communicate with internal components and services, including the D-Bus service. With the emulated firewall in place, the protective shell effectively blocks any communication attempts made by external devices to the internal components or services, significantly mitigating the failure mode of controlling automotive features from external devices and aligning with the principles of a zero-trust architecture essential for mixed critical systems.

In the event that an attacker successfully infiltrates the infotainment system and attempts to spread and take control of core vehicle components, the protective shell is equipped with mitigation mechanisms designed to limit the attacker's capabilities and isolate them from further spreading. By utilising advanced anomaly detection based on behaviour model training, the internal processes and component communications are continuously monitored for suspicious activities. Any abnormal behaviour deviating from the trained baseline, such as the attacker sending out malicious messages, is promptly identified by the protective shell, triggering an alert to the driver and initiating protective actions. Therefore, the protective shell provides fundamental defence against attackers who have infiltrated the Uconnect system and attempts to slow down their ongoing cyber-attack.
Furthermore, the code-signing capability of the protective shell enforces integrity checks

upon the protective shells and the CPS's code base, mitigating failure modes where attackers could easily update underlying components with manipulated firmware. In the context of the analysed security incident, it prevents the implementation of a backdoor that would allow the sending of arbitrary CAN messages. Additionally, it also addresses potential malicious intent to update a component with non-functional firmware, which could render its functionality useless making it unavailable. Such unauthenticated updates are prevented, thus limiting the attackers to exploit already present vulnerabilities in the system.

Further failure modes can be mitigated through a thorough analysis of the operation logs provided by the protective shell whenever anomalies are detected or intervention becomes necessary. Together with its alerting capabilities, this aids Harman in continuously improving their Uconnect system by effectively eliminating newly detected vulnerabilities.

Identification of New Vulnerabilities The implementation of the brute-force attack detection and prevention mechanisms may, in rare cases, lead to the blocking of normal users from authenticating. Two reasons for this could occur. First, during an active brute-force attack, the protective shell forwards only a limited number of authentication requests per minute to the Wi-Fi module and discards the rest. As a result, if a valid user attempts to authenticate during such an attack, their authentication request might get discarded, leading to a situation similar to denial of service. Second, if a user initiates too many authentication attempts with the wrong password, the protective shell may temporarily block their MAC address. In such cases, after a short waiting period, the user should be able to make another attempt at authenticating with the correct password.
Another failure mode that might arise from the protective shell is falsely detecting anomalies, such as suspicious messages on the CAN-Bus, and subsequently blocking them. Although this situation is unlikely, the consequences could be severe, especially if the affected messages are safety-critical. To address this, the protective shell must be trained using suitable machine learning algorithms based on a vast amount of trustworthy operational data and log files. The resulting behaviour model must be thoroughly evaluated by domain experts. Additionally, the protective shell must consider the severity of each message concerning the system and may choose not to intercept a suspicious CAN message if the message does not compromise the overall safety and security of the system.

The last new weakness identified in this evaluation are the consequences a more complex update mechanisms brings due to the code-signing capabilities of the protective shell. While such security mechanisms are widely accepted in the industry, inexperienced manufacturers may encounter difficulties in signing new code versions and rolling out updates for their CPS. This could lead to increased waiting times for important safety- or security-critical updates to reach their customers, providing attackers with an extended time-frame during which vulnerabilities remain exploitable or failures to cause larger amount of preventable accidents.

Conclusion The evaluated protective shell incorporates multiple mitigation mechanisms, significantly enhancing the system's trustworthiness by effectively addressing most of the vulnerabilities found in the Uconnect infotainment system. Notably, the protective shell introduces a few new failure modes; nevertheless, their risks remain within acceptable ranges and are considerably lower than the failure modes that have been successfully remediated. Overall, the implementation of the protective shell reduces the residual risk to an acceptable level.

Considering the evaluation, it is highly plausible that the protective shell successfully prevents the attack pattern presented by the security experts Miller and Valasek and contributes further to the resilience of the Uconnect system to cyber-attacks. The thorough assessment of the protective shell's features and their impact on avoiding failure modes underscores the importance of such security measures in CPS, particularly in critical domains like the automotive industry.

5.6 Ukrainian Power Grid Cyber-Attack

5.6.1 Introduction

The Ukrainian Power Grid attacks refer to a series of coordinated cyber-attacks that targeted the country's power grid in 2015 and 2016. The analysis focus will be on the cyber-attack executed on the 23rd of December 2015, as it exhibited noteworthy similarities to the incident in 2016.
The consequences of this attack were far-reaching, resulting in a widespread power outage across multiple regions and impacting over 225,000 customers. It marked the first large scale cyber-attack on electric infrastructure to be recorded. The attack focused on three electric power distribution companies and took out more than 30 of their substations. The responsive companies acted promptly though diligently and managed to manually restore power to their customers within a range of 1 to 6 hours. Given that the attack occurred during winter, there was immense time pressure to restore power promptly, as any delay would risk freezing water pipes and pumps, thus permanently damaging household components which would lead to prolonged heat outages.
Nevertheless, it took several months for the electric companies to fully recover their digital infrastructure and resume normal operations.
The overall damage caused by these attacks is estimated to be in the millions of dollars. The Ukrainian government has accused Russian state-sponsored hackers of carrying out these attacks, while Russia has vehemently denied any involvement. It is widely believed that the objective of these attacks was to destabilise the Ukrainian government and economy.
This incident stands as one of the most significant instances of cyber-attacks on critical infrastructure, serving as a warning of the potential risks posed by cyber-attacks on power grids and other crucial infrastructure. It also served as a wake-up call for other nations to reassess the security of their critical infrastructure and bolster their cyber defences.

5.6.2 Vulnerabilities in the critical Infrastructure

The following attack pattern and vulnerabilities were gathered from reports by the International Society of Automation (ISA) [28] and the U.S. Homeland Security NCCIC [29].

Insufficient Social Engineering Protection Training Eight months prior to executing the payload, the attacker initiated the preparation and reconnaissance phase. To gain initial access to the IT network, they launched a spear-phishing attack aimed at employees on the business side of the power supplier. By exploiting social engineering techniques, they manipulated inadequately trained employees into opening malicious email attachments in the form of Microsoft Office documents, which instructed them to enable macros. Once the macros were enabled, the employees' computers were infected with the Black Energy 3 Trojan.

Black Energy 3 is a malware suite equipped with powerful components such as the KillDisk wiper and utilities for keystroke logging. The attackers utilised the Trojan to conduct network scans, propagate to new devices within the local network, and extract credentials, including those from Windows domain controllers. They gathered intelligence on further device vulnerabilities, familiarised themselves with the network structure, installed additional malware, and established backdoors.

No Network Monitoring During the post-incident analysis, it became evident that the attackers could have been stopped in the reconnaissance phase if there was proper network

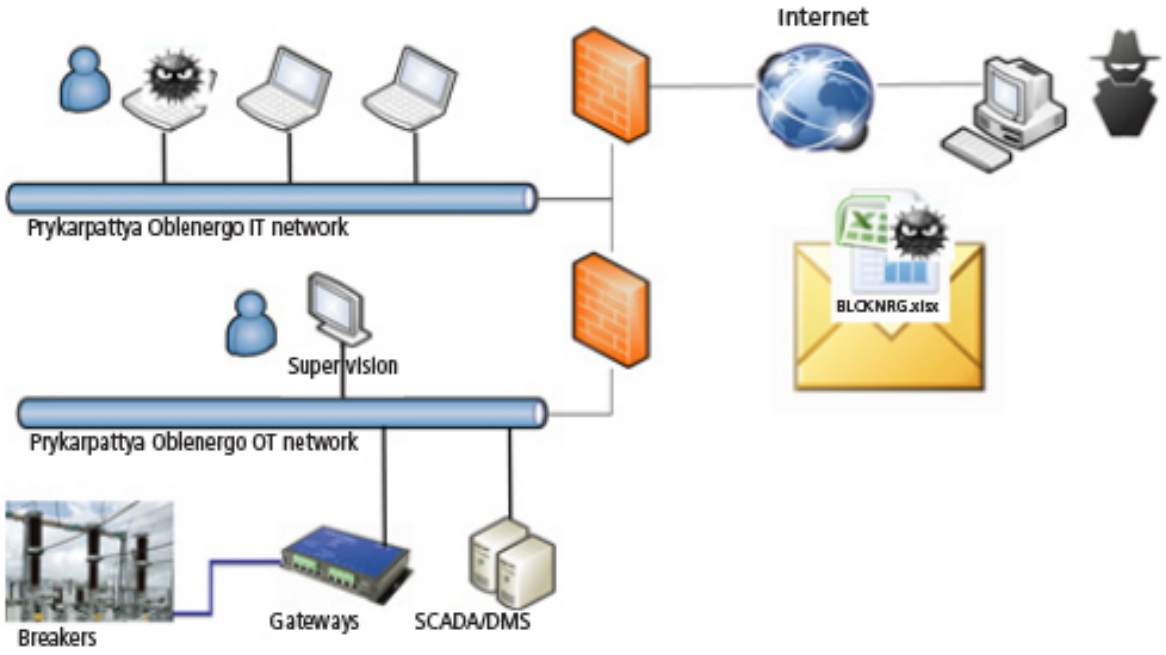


Figure 5.4: Ukrainian Power Grid Attack - Initial Intrusion IT network [28]

supervision. The absence of network monitoring for suspicious packets made the execution of extensive vulnerability scans and network mapping possible. Otherwise, the attackers would likely not have been stayed undetected crawling through the network for months. Nowadays, intrusion detection systems (IDS) or intrusion prevention systems (IPS) would automatically identify large-scale network scanning operation packets as suspicious amidst a sea of "good" packets. They would promptly notify administrators or might as well initiate protective actions. Nevertheless, even manual monitoring of network traffic without deep analysis or behaviour-based anomaly detection could have detected the intelligence gathering conducted by the attackers in this case.

Insecure Network Segmentation Companies tend to take cybersecurity in their IT network not as serious, assuming that their operation technology (OT) network is completely segmented from it. This false sense of security is often reinforced by the term "Air-Gap". As illustrated in figure 5.4 above, there was indeed a firewall separating the IT network from the (control) OT network. However, employees were granted remote access to devices in the OT network through a VPN for maintenance purposes.
During the reconnaissance phase, the attacker managed to infiltrate numerous devices in the IT network and discovered on one device an open SSH tunnel to the OT network. They began recording keystrokes to obtain important information. They were able to collect VPN's credentials used by the employees to remotely control their SCADA systems. During non-office hours, they established the SSH tunnel to the OT network and started to infect the first devices there.

Effective network segmentation offers enhanced security against cyber-attacks only if it cannot be easily bypassed. The credentials used were either found directly on the local machines or could be gathered by keyboard stroke recording. Ideally, a strong segmentation or the operation of two distinct networks would be optimal. However, if remote operation from employees in the IT network is necessary, robust multifactor authentication should be implemented, and strict remote access policies must be enforced.

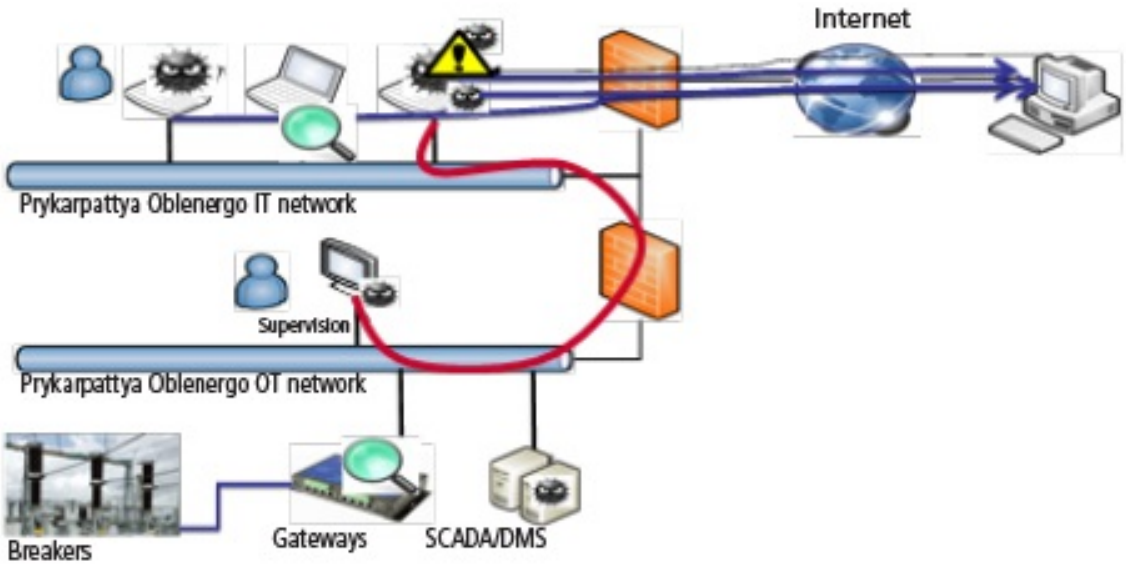


Figure 5.5: Ukrainian Power Grid Attack - Spread to OT network [28]

Bad Firmware Updates During the execution phase of the cyberattack on the 23rd of December, the attackers gained remote access to the SCADA systems and deliberately opened several circuit breakers at over 30 substations. This caused a power supply disruption for more than 225,000 households. In order to hinder swift restoration efforts by employees remotely closing the breakers, the attackers initially locked them out and changed their passwords. Furthermore, after deploying their malicious payload, they proceeded to delete numerous files stored on workstations and servers using the KillDisk wiper. This served the purpose of obfuscating the attackers' tracks and impending the recovery phase of the affected companies.
Additionally, the attackers rendered the serial-to-ethernet converters useless by remotely updating their firmware with non-functional versions, utilising credentials obtained during the preparation phase. These converters play a vital role in providing industrial machinery with an interface for Ethernet communication, converting various RS standards. By incapacitating these converters with "faulty" firmware, the electric energy supplier companies suffered a complete operational loss of their grid distribution systems.
Lastly, they disabled the uninterruptible power supply (UPS) located at the control centre by also overwriting the used firmware with a non-functional one. Consequently, the control centre also experienced blackouts as the UPS failed to provide backup power.

The affected companies need to evaluate the necessity of remote firmware updates for such low-level components and determine whether the additional risk outweighs the benefits. Given that the functionalities of ethernet-to-serial converters and UPS are typically static and do not require frequent updates, locally performed firmware updates would be suitable in most scenarios.
However, allowing unsigned firmware to run on these devices should not be a subject of debate. Manufacturers of critical infrastructure components should consistently provide digitally signed firmware to mitigate the risks associated with the use of potentially malicious or counterfeit firmware.

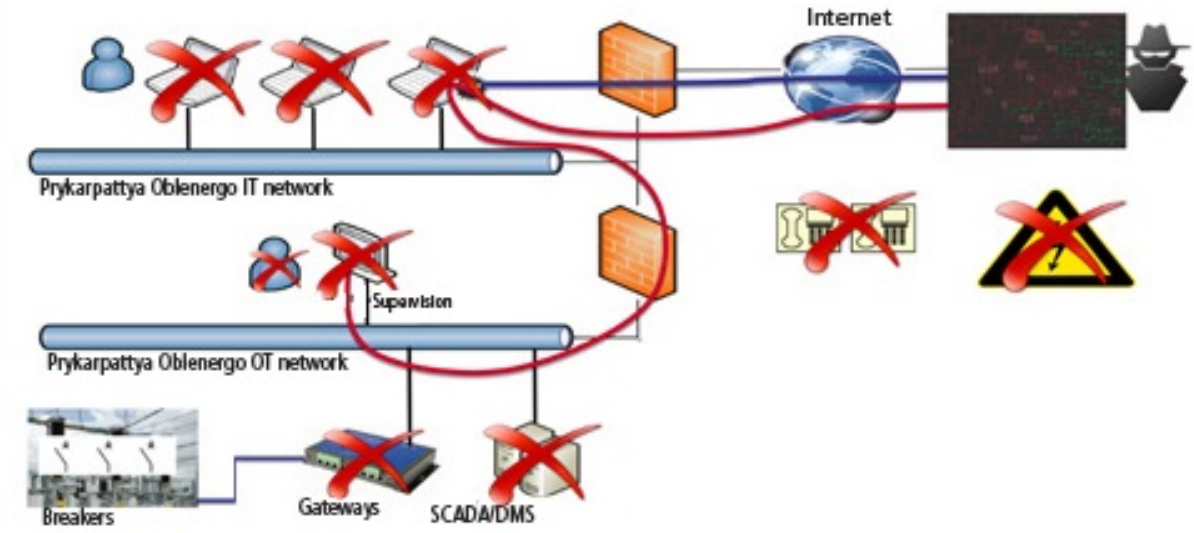


Figure 5.6: Ukrainian Power Grid Attack - Successful Payload Execution [28]

Supply Chain Vulnerability One potential factor that may have contributed to the success of the cyber-attack is the historical relationship between Ukraine and Russia. However, it is important to note that limited information is available on this matter, and thus, it should be considered as a possibility rather than a definitive fact. Numerous authorities from different countries have attributed the cyber-attack to Russia. It is noteworthy that significant portions of the Ukrainian power grid were built before the collapse of the Soviet Union and subsequently upgraded using components manufactured in Russia. Consequently, the Russian attackers possessed insider knowledge of the system's operation, were familiar with its overall architecture, and may have been able to identify initial vulnerabilities in their own components. Additionally, the attackers were already acquainted with the software used to control the power grid, which suggests the possibility that they had received prior training to familiarise themselves with its functionalities.
The fact that the attackers had extensive knowledge of the power grid infrastructure architecture and were familiar with the software used does not inherently constitute a vulnerability. As stated by Kerckhoffs's principle, "There should be no security assumptions based on system obscurity." However, there is always the potential for suppliers in the critical infrastructure supply chain to ship components with intentionally or unintentionally incorporated vulnerabilities or the components run with outdated and unpatched software.

5.6.3 Specific Protective Shell Mitigation Mechanisms

The specific protective shell developed for the SCADA systems controlling the Ukraine power grid breakers incorporates several core mitigation mechanisms to address the vulnerabilities that led to the security incident previously analysed. The purpose of this protective shell is to safeguard the CPS responsible for controlling critical power infrastructure and delivering services to thousands of customers and corporations. In this section, key mitigation mechanisms are presented, defining the main characteristics of this protective shell.

Reconnaissance Detection As previously analysed, the attacker remained undetected in the power distributor's networks for months, engaging in network device crawling and scan-

ning. While the protective shell does not encompass the entire IT/OT network and thus cannot replace a rigorous network monitoring solution, it effectively covers the SCADA system and its components. This allows for the detection of suspicious scanning activities within the CPS, providing a reconnaissance detection mitigation mechanism. During the attacker's planning and information-gathering phase before executing the cyber-attack and delivering its payload, they must have left traces of scanning devices within the OT network, including the SCADA system and its underlying components. Although these traces may be non-persistent and only manifest as packets in the SCADA communication, the protective shell can detect those suspicious messages and alert personnel to investigate further. Consequently, the attackers are placed under time pressure upon detection, or they can only execute minimal scanning operations, trying to avoid detection large scale network mapping scans would have. This mitigation capability is achieved in cooperation with the next mitigation mechanism, the behaviour model training.

Behaviour Model Training Another critical mitigation mechanism involves the training of a behaviour model used for advanced anomaly detection, similar to the other specific protective shells discussed earlier. This training is based on a vast amount of operational data and log files generated by the SCADA systems and their underlying supervising components. The established behaviour baseline encompasses normal operation conditions, which is trained using suitable machine learning algorithms. One use case where advanced anomaly detection proves crucial for CPS security is the reconnaissance detection mentioned earlier. By employing anomaly detection, the protective shell can identify increasing amounts of analysis data and scanning activities in the SCADA systems, triggering alerts for personnel.

Another promising use case for anomaly detection based on the behaviour model is sophisticated Identity and Access Management (IAM). The behaviour model would contain various intrinsic knowledge about login attributes in addition to the username and password. For example, an employee's login pattern is relatively static, involving logins during workdays at consistent times, from the same IP address (or at least the same region to which the IP address belongs), and using the same computer. Minor deviations from this pattern, such as logging in four hours late due to an appointment or using a new computer, may be permitted. However, during the security incident analysis which involved using stolen credentials, abnormal behaviour was clearly evident. The attacker used the stolen credentials to log in at out-of-office times and from different computers with external IP addresses. While the protective shell does not cover the employees' Windows computers, it can effectively detect abnormal behaviours, such as the attacker's actions within the OT network, initiating SCADA systems operations during out-of-office hours from previously unknown IP addresses. This would be a very obvious abnormal behaviour to detect by the protective shell and initialising protective actions, alerting the employees and blocking the credentials that performed suspicious actions.
IAM features were formerly defined by static rule-sets and specifications. However, since the login and authentication process can gather many more secondary attributes linked to the user, there is a pursuit to use more sophisticated evaluations to determine if a user should be given access or not. The whole range of data sets can't be easily evaluated by static rules, therefore the behaviour model anomaly detection seems suitable.

The third interesting use case for advanced anomaly detection in this protective shell is monitoring user actions. The protective shell gives each user session has a limited number of actions they can initiate in the SCADA system. Once this limit is reached, the user must either re-authenticate and/or obtain approval from a second user. This tactic, commonly used in API tokens, limits their potential for damage if they fall into the wrong hands. For

instance, if a user makes several settings changes and initiates numerous API calls within a short period, the user is prompted to re-authenticate, sometimes with different authentication factors, to restore "balance" to their token and regain the system's trust.
This technique is adopted by the protective shell, which defines a non-limiting amount of actions provided to user accounts for their day-to-day activities in the SCADA systems. The protective shell calculates this during the behaviour model training phase. For example, the model might incorporate practices such as "allowing one user to open a maximum of five circuit breakers in the power grid every hour", as the protective shell has learned that no single user would normally be restricted by this rule. However, such a rule would slow down power grid cyber-attacks, such as the one analysed previously, which aimed to open as many circuit breakers as fast as possible. All user actions in the SCADA system are continuously monitored and evaluated by the protective shell to determine if this user exhibits any abnormal behaviour.
An effective mitigation mechanism fitting this situation, that the protective shell does not provide is multifactor authentication, either during initial authentication or for re-validating the current session with added trust.

Code Integrity Another critical mitigation mechanism employed by the protective shell, to safeguard the SCADA system with underlying components, is code and firmware signing capabilities. As already explained in section 5.5.3, this technique ensures that execution is limited to trusted code only.
In this case, it effectively prevents attackers from manipulating the code base of the embraced CPS. For instance, if attackers attempt to update the firmware of the serial-to-ethernet converters with a non-trusted version, the protective shell's automatic integrity checks would fail, and the update request would be intercepted. Therefore, updates with "faulty" firmware, which could render the components nonfunctional as exploited in the analysed security incident, would be prevented.
Unfortunately, the protective shell does not embrace the UPS located at the control centre, which also became nonfunctional due to manipulated firmware, and therefore has no influence on providing security mechanisms for them.

Alerting and Logging Similar to the mitigation mechanisms presented in other specific protective shells, the SCADA system and components safeguarded by the protective shell also require alerting and logging capabilities for likewise reasons. Particularly in this scenario, where a potential cyber-attack affects many other devices in both the IT and OT network, timely informing employees whenever an anomaly is detected or when the protective shell initiates protective and corrective actions is crucial. This aids in promptly initiating and guiding the security incident response by experts.
Furthermore, detailed operation logs aid in the forensic analysis of security incidents, enabling the discovery of previously unknown vulnerabilities and improving overall system reliability and trustworthiness through future updates.

5.6.4 Protective Shell Evaluation

This section presents a rigorous evaluation of whether the "Ukraine Power Grid" security incident could have been prevented through the implementation of a specific protective shell equipped with the previously described mitigation mechanisms. The primary objective of the protective shell is to safeguard the SCADA system and underlying components from successful exploitation of the identified and analysed vulnerabilities. Therefore, the protective shell must effectively address those vulnerabilities in the original critical infrastructure design.

The evaluation seeks to assess the effectiveness of the protective shell's capabilities in mitigating the identified vulnerabilities through a series of monitoring and validation processes, ultimately reducing the residual risk associated with potential incidents. Additionally, the evaluation investigates whether the implementation of the protective shell introduced any new vulnerabilities that could compromise the overall security and reliability of the system.

Remediation of Old Attack Vectors In this case, the protective shell has limited influence in remediating some attack vectors along the attack path. Notably, the protective shell cannot improve the insufficient social engineering protection training described in the analysis, as it falls outside its sphere of influence. Similarly, due to its location embracing only a small subset of the OT network, it cannot fully address the network monitoring problem, which sophisticated IDS and IPS would typically cover for the entire IT and OT network. Consequently, the protective shell cannot directly improve the insecure network segmentation configuration or implement security mechanisms to enhance segmentation barriers, as described in the analysis. Lastly, the protective shell has no influence on the underlying CPS architecture, including which components get implemented from which supplier. Therefore, the possible supply chain vulnerability discovered in the analysis is out-of-scope for the protective shell.

However, the protective shell can implement core mitigation mechanisms that address a subset of the identified failure modes. For instance, the implemented reconnaissance detection for SCADA controlled systems allows for early detection of cyber-attacks during the information gathering phase. Therefore, the attacker might not stay undetected for months during their reconnaissance, rather could be detected as soon as they start scans on the SCADA systems. This serves as an intrusion detection mechanism further along the attack path, even if the intrusion point is located outside the protective shell's influence.

Additionally, the advanced anomaly detection based on the trained behaviour baseline provides the protective shell with capabilities to remediate several failure modes. The IAM can be improved using identity attributes for authentication processes previously not utilised. This could identify malicious abnormal user activity if the attacker would log in with stolen credentials from unknown IP addresses during out-of-office hours. Moreover, the protective shell closely monitors user actions, limiting the number of changes one user session can make to the SCADA system and its components, before they need to re-authenticate and/or get approved by a second user, the protective shell drastically limits the damage potential one user can expose to the system. Therefore, during the payload delivery phase, the protective shell would slow down the cyber-attack by placing hurdles in the attacker's way like temporarily revoking user rights to open any more circuit breakers for a specified period.

Another valuable mitigation mechanism implemented by the protective shell is the code and firmware signing process. By enforcing this process, the protective shell remediates the failure mode where attackers could remotely initiate unsigned firmware updates to critical infrastructure components in the SCADA systems. With this mitigation mechanisms, the protective shell first validates that the new firmware update is trusted and not manipulated before allowing the update request to be forwarded to the components. In this case, the serial-to-ethernet converters would not have been rendered unusable with the malicious firmware blocked by the protective shell, enabling the power grid employees to communicate with the power grids critical components remotely. Consequently, the circuit breakers could have been closed directly from accessing the OT network and not needing to send workers to the breakers to manually close them, which would reduce the mean-time-to-recovery (MTTR) significantly.

Identification of New Vulnerabilities Despite its advantages, the protective shell introduces a few new risks that need to be considered. In emergency situations, such as cable fires or floods, where rapid opening of many circuit breakers is necessary, the protective shell may misinterpret user actions as abnormal and restrict access, potentially slowing down the emergency response. However, given the cooperative nature of controlling critical infrastructure components, it is unlikely that only one user would have remote access in such situations or trying to close them alone, rather several users would work together limiting the damage amount.
Furthermore, the protective shell's detection of abnormal login patterns might occasionally block non-malicious users from accessing the system, leading to reduced user experience. Although this is not safety- or security-critical, it enhances the overall trustworthiness of the system.
Additionally, the protective shell's false detection of anomalies and subsequent triggering of alerts could lead to significant work efforts in containing fictional cyber-attacks. Analysing log files and identifying false positives may delay other important processes, such as development and expansion projects, prioritising cyber incident response.
The last new weakness identified in this evaluation are the consequences a more complex update mechanisms brings due to the enforced code-signing process by the protective shell. While such security mechanisms are widely accepted in the industry, inexperienced manufacturers may encounter difficulties in signing new code versions and rolling out updates for their CPS. This could lead to increased waiting times for important safety- or security-critical updates to reach their customers, providing attackers with an extended time-frame during which vulnerabilities remain exploitable or failures to cause numerous amounts of preventable accidents.

Conclusion This evaluation showed that the protective shell alone did not completely eliminate the identified vulnerabilities as desired, given the nature of cyber-attacks that can span beyond the protective shell's specific sphere of influence. Nevertheless, the protective shell still has a considerable effect in preventing catastrophic outcomes especially targeting its embraced CPS, as analysed before.
The evaluated protective shell incorporates multiple mitigation mechanisms, significantly enhancing the system's trustworthiness by effectively addressing core vulnerabilities found within the SCADA system.
Notably, the protective shell introduces a few new failure modes; nevertheless, their risks remain within acceptable ranges and are considerably lower than the failure modes that have been successfully remediated. While the implementation of the protective shell may not reduce the residual risk of the power distributors' networks to an acceptable level, it effectively safeguards the SCADA system and its components in cases of cyber-attacks.
Considering the evaluation, it is highly plausible that the protective shell successfully interferes with the attack chain when the attacker reaches the SCADA system, blocking and slowing down the cyber-attack. The thorough assessment of the protective shell's features and their impact on avoiding failure modes underscores the importance of such security measures in critical infrastructure domains.

5.7 Airbus A400M FADEC

5.7.1 Introduction

The Airbus A400M is a military transport carrier designed for transporting cargo and troops. On May 9, 2015, a safety accident resulted in a fatal crash of an A400M aircraft near Seville, Spain.
On that day, a pre-delivery check flight for the Turkish Air Force was scheduled. Shortly after takeoff, the crew transmitted an SOS signal, requesting an emergency landing due to technical difficulties. While attempting an emergency landing in a field, the aircraft collided with a high-voltage line and caught fire. The crash claimed the lives of four Airbus Defence and Space personnel and left two crew members injured.

The Technical Investigation Commission on Military Aircraft Accidents (CITAAM) analysed the flight data obtained from the two "black boxes" - the flight data recorder (DFDR) and the cockpit voice recorder (CVR). Their findings revealed that three of the four turboprop engines did not respond to the pilots' input, with only one functioning engine providing insufficient power to sustain the aircraft airborne.
During the investigation, it was discovered that a software vulnerability existed in the Full Authority Digital Engine Control (FADEC) system. The FADEC system controls the engines of the A400M and manages their performance. During the software installation, crucial torque calibration parameter data was inadvertently erased from the three malfunctioning engines. Consequently, the FADEC system was unable to interpret how to operate the engines, resulting in the engines being locked at maximum power after takeoff.
As a result, the A400M rapidly ascended while accelerating. The crew was contacted by tower controllers when the aircraft reached a maximum altitude of 1725 feet, instructing them to maintain the predefined altitude of 1500 feet. The crew attempted to comply with the order by reducing thrust, unaware that the faulty engines could only operate at full power or idle. Unfortunately, after reducing the thrust, the engines remained locked at idle, leaving only one engine operational. Within seconds, the plane descended and crashed into the field.

Some safety experts view this crash as an illustration of how failures, although rare, can arise in increasingly complex aircraft systems when multiple weaknesses align to create a significant risk [30]. The main vulnerability in this instance was attributed to inadequate testing and validation of the FADEC software. This accident underscores the critical importance of conducting thorough testing and validation of software in safety-critical systems such as aircraft engines.

5.7.2 Vulnerabilities within CPS

Insufficient Mitigation for a Known Issue CITAAM revealed that the engine developer and manufacturer, Euro Prop International (EPI), had already notified both Airbus and the European Aviation Safety Agency (EASA) about a software installation error that could result in the loss of engine data. According to a safety document reviewed by Reuters, under the design of the A400M, pilots would only receive their first warning of the engine data problem when the aircraft reached an altitude of 400 feet. There is no cockpit alert available on the ground [31]. CITAAM concluded that the mitigation measures implemented by Airbus and EASA were inadequate [32].

It is likely that the Airbus management did not prioritise addressing this issue due to time-to-market pressures, declining profit margins, and the need to fulfil their military contracts.

The A400M was developed for Spain, Belgium, Britain, France, Germany, Luxembourg, and Turkey, and has been plagued by delays and cost overruns, which have significantly surpassed the initial budget of 20 billion euros [30]. As early as 2009, the A400M project was already 5 billion euros over budget, 3 to 4 years behind schedule, and costing Airbus between 1 billion and 1.5 billion euros annually [33]. Moreover, a PricewaterhouseCoopers audit projects a budget overrun of approximately 11.2 billion euros upon the completion of the project.

No Software Integrity Checks Before every system start, it is essential to conduct software and data integrity checks to ensure that the underlying code base is complete and unaltered. Otherwise, the start-up must be interrupted. This is particularly crucial for safety-critical systems like FADEC before the aircraft can proceed with its pre-takeoff preparations.
In addition to cockpit alarms indicating failed software verification, the engines should remain idle until they receive the appropriate data. The only reason the A400M was able to take off without torque calibration parameter data in three of its engines was because they could be powered at full throttle.

Responsibility Dispute According to the CITAAM investigators, there is a disagreement between Airbus and EPI regarding who was responsible for installing the engine software. CITAAM concluded that the lack of coordination between Airbus and EPI, among other factors, contributed to the safety accident.
The software was installed by Airbus workers using the plane maker's systems, but EPI argues that it should have been loaded by their own staff using EPI systems, as they possess the necessary civil authorisation.
Airbus claims that since the A400M is a military aircraft, their personnel were responsible for uploading the software [32]. This highlights the regulatory confusion surrounding the civil and military jurisdiction over the aircraft at the time of the accident. The A400M is a rare hybrid, being a military plane with European civil certification. Spanish officials have supported Airbus by asserting that the assembly line is a defence facility and not subject to civil regulations [30]. However, this dispute should have been resolved prior to the final assembly of the aircraft.

Marwan Lahoud, the Chief Strategy Officer of Airbus at the time, informed the German press that "The black boxes attest… that there are no structural defects, but we have a serious quality problem in the final assembly". This concisely summarises the underlying causes of the safety accident.

5.7.3 Specific Protective Shell Mitigation Mechanisms

The specific protective shell developed for the Airbus A400M FADEC system incorporates several core mitigation mechanisms to address the vulnerabilities that led to the previously analysed safety accident. The primary objective of this protective shell is to safeguard the CPS from failures that could impede its ability to deliver the safety-critical feature of engine control. In the following section, those key mitigation mechanisms are presented, which build the main characteristics of this protective shell.

Hardened Process for Software Installation In this academic scenario, benefiting from a retrospective view, most of the vulnerabilities discovered and mitigated within the protective shell were previously unknown before the accident/incident. However, it is important to note that in this special case, the vulnerability of an erroneous software process was already

known and had been brought to attention by EPI. Consequently, Airbus should have proactively implemented similar mitigation mechanisms, as presented below, before the safety accident occurred.
The protective shell provides an isolated installation environment for the FADEC software, wherein the installation process is encapsulated to access only the FADEC software files. Meanwhile, the configuration files per engine are kept separate from this encapsulation to preserve their integrity between software versions. If reconfiguration becomes necessary, the protective shell offers an additional, separate process for conducting the adjustments.
Moreover, considering the information published by EPI regarding the conditions under which software installation errors might occur and how to detect them, the protective shell can closely monitor the installation process and promptly notify the operator if any anomalies are detected. To ensure the successful installation of the FADEC software, the protective shell could adopt a process similar to Continuous Integration / Continuous Development (CI/CD) pipelines commonly used in the industry. Such a process involves conducting tests and further analysis to verify the integrity of the installation.

Code Integrity Another critical mitigation mechanism implemented in the protective shell is the use of software integrity checks before allowing system start-up. While this technique was previously employed in protective shells to prevent malicious users from manipulating the code base, in this scenario, it serves to verify the completeness, unaltered state, and trustworthiness of the code base containing the FADEC software and configuration files. The mechanisms are identical to the process already described in the specific protective shell for the Uconnect system 5.5.3.
During system start-up, the protective shell calculates the hash of its encompassed CPS code base and compares it with a predefined, trusted hash stored within its own (signed) code. Even a slight deviation in the code results in a complete hash value change, indicating the presence of a non-trusted version of the CPS software. This capability is instrumental in preventing start-up with missing torque calibration parameter data for one or several engines, which was one of the main contributing factors to the Airbus A400M crash. With this implemented mitigation mechanism, the protective shell would intercept the start-up process, preventing any functionality provided by the CPS until the issues are resolved.

Alerting and Logging Similar to other specific protective shells, the protective shell for the FADEC system requires alerting and logging capabilities for similar reasons. While the other mitigation mechanisms enable the protective shell to detect and correct failure modes, it is equally important to inform the CPS operator and manufacturer about the detections and interventions.
Alerts serve to notify the operator of detected anomalies and assist them in locating and resolving issues promptly. For example, if the engine calibration data is missing, a meaningful cockpit alert must be displayed while the aircraft is on the ground. Additionally, detailed operation logs are instrumental for the manufacturer in discovering previously unknown vulnerabilities and improving the overall system reliability and trustworthiness through future updates. These logs can provide valuable insights into the system's performance and aid in iterative refinement to enhance safety and security.

5.7.4 Protective Shell Evaluation

This section presents a rigorous evaluation of whether the Airbus A400M accident could have been prevented through the implementation of a specific protective shell equipped with the previously described mitigation mechanisms. The primary objective of the protective shell is to safeguard the FADEC system from faults related to engine failure, which was

the contributing factor leading to the safety accident. Therefore, the protective shell must effectively address the vulnerabilities in the original FADEC design that were uncovered during the accident analysis.
The evaluation aims to assess the effectiveness of the protective shell's capabilities in mitigating the identified vulnerabilities through a series of monitoring and validation processes, ultimately reducing the residual risk associated with potential failures. Additionally, the evaluation investigates whether the implementation of the protective shell introduced any new vulnerabilities that could compromise the overall safety and reliability of the system.

Remediation of Old Failure Modes The protective shell is equipped with several highly effective mitigation mechanisms to address the failure modes related to uncontrollable engine turbines.
Firstly, the protective shell provides a hardened installation process for the FADEC software on each turbine to prevent installation errors and their side effects, such as the inadvertent deletion of torque calibration data. If an error occurs during the monitored installation process, the protective shell promptly alerts the operator and provides detailed information about the cause of the error. Furthermore, in cases where the software is installed without errors, the protective shell triggers the next CICD step to test the installation at hand. These mitigation mechanisms effectively prevent the failure mode of missing calibration data after an erroneous installation process, thereby reducing the risk of a turbine running with an incomplete FADEC software version.

Secondly, the protective shell provides code signing capabilities and integrity checks for its own and the CPS's code base. This ensures that the Airbus A400M turbines cannot operate without a verified and complete FADEC code version. In this way, the protective shell intercepts engine start off if the code integrity checks fail, ultimately preventing lift-off. This mitigation mechanism addresses the failure mode of operating the Airbus A400M with an unverified and incomplete FADEC code version.

Identification of New Vulnerabilities With the implementation of a more sophisticated and secure installation process and the inclusion of code integrity checks, the evaluation did not identify any new vulnerabilities worth mentioning. The protective shell effectively resolves known issues while not introducing any obvious drawbacks.

Conclusion The evaluated protective shell incorporates multiple mitigation mechanisms, significantly enhancing the system's trustworthiness by effectively addressing vulnerabilities found in the FADEC CPS. Importantly, during the evaluation, no new failure modes introduced by the protective shell were identified, indicating that Airbus could have implemented similar mitigation mechanisms without overly risking adding new vulnerabilities to the system.// Considering the thorough assessment of the protective shell's features and their impact on avoiding failure modes, it is highly plausible that the Airbus A400M crash associated with engine failures could have been prevented with the suggested implementation of this protective shell, thus significantly reducing the residual risk. The comprehensive evaluation underscores the importance of such safety measures in CPS, particularly in critical domains like aviation.

5.8 Similarities between Specific Protective Shells

This section provides a summary of the different mitigation mechanisms employed in the presented specific protective shells. Each mitigation mechanism can be classified under one of the following category umbrella terms. However, it is important to note that *not every specific protective shell utilises mitigation mechanisms from every category*, as their applicability depends on the security or safety improvement they provide to the embraced CPS. Consequently, different protective shells may employ distinct mitigation mechanisms if the general architecture permits it. The following categories were derived by analysing overlapping mitigation mechanisms that were frequently employed, summarising the common characteristics exhibited by the different protective shells.
After presenting the list of mitigation mechanism types, they will be briefly explained. This section draws upon the experiences gained from analysing security incidents and safety accidents while developing and evaluating the specific protective shells, which was the main focus of this chapter. By summarising the observed similarities and carrying them forward to the next chapters, this thesis aims to formulate more general architecture and design principles for protective shells in a generic framework.

5.8.1 Mitigation Mechanisms Categories

1. Alert and Logging
    a) Meaningful Alerts
    b) Detailed Logs
2. Code Base Integrity Checks
    a) Firmware/Software Signing
    b) Data Completeness
3. Sensor Readings / Application Signals
    a) Reevaluation (Range Validation)
    b) Cross-Validation
    c) Time-sequential Analysis (Retrospective Data Evaluation)
    d) Supplementary Sensors
4. Monitor
    a) Communication
    b) Processes
    c) IAM
5. Behaviour Model Training

5.8.2 Explanation

1a. Meaningful Alerts In all the different protective shells discussed, the *capability to provide meaningful alerts to the CPS operator was consistently necessary*. These alerts were either displayed directly on an integrated screen, such as in aeroplanes or vehicles, or forwarded to an external monitoring solution, as seen in SCADA systems. The purpose of these alerts was to notify the operator whenever the protective shell identified an anomaly, when a validation process was unsuccessful, or when corrective and protective actions needed to be initiated to intervene with the CPS's intended actions.
The alerts as real-time status information helps the operator to investigate and take manual actions in response to impending failure modes. Immediate alerts are crucial because abnormal behaviour might otherwise only be detected upon analysing operation logs, and therefore potentially not detected at all. If the protective shell did not intervene, it could lead to security incidents or safety accidents within the underlying CPS. Furthermore, such alerts put pressure on CPS manufacturers to analyse the cause of the alert and mitigate any found vulnerabilities or adjust the protective shell. By doing so, either the CPS or its safeguarding protective shell gets enhanced.

1b. Detailed Logs *Detailed logs of protective shell operations were also found to be necessary in all the presented sets of mitigation mechanisms* providing means for forensic analysis. These logs primarily aid the protective shell manufacturer in analysing alerts reported by operators or during post-accident / incident processes. Having detailed logs is crucial for finding the root cause after catastrophic CPS behaviour, understanding it, and being able to mitigate it effectively. In instances where a false alarm occurs from the protective shell, adjusting the detection mechanisms to prevent further false alarms of that type may be necessary. Detailed logs provide manufacturers with a better understanding of the often complex CPS, enabling them to eliminate newly detected vulnerabilities or improve the protective shell to better safeguard the CPS.

2a. Firmware/Software Signing While not every specific protective shell employed code signing as a mitigation mechanism, those that did found significant benefits in enhancing the system's overall trustworthiness. Code signing relies on a trusted entity, such as the manufacturer, to sign the protective shell's code base. This code base also includes knowledge about the "fingerprint" or hash of the embraced CPS's code base. During the CPS startup process, the protective shell verifies if the embraced CPS code version is valid, unaltered and trusted. If the checks are unsuccessful, the protective shell intercepts the startup process, preventing any functionality provided by the CPS until the code version reverts to a known one, thereby preventing undesired behaviour.
This mechanism is also applicable when updating embraced components in the CPS during run-time, which always requires an update to the protective shell's trusted hash values before permitting the update. By utilising this mitigation mechanism, the *protective shell safeguards itself and the embraced CPS from malicious modifications*, such as backdoor implementations or non-functional firmware rendering the component useless.

2b. Data Completeness The code base integrity checks offer protective shells not only the capacity to safeguard against malicious intentions, but also to *abstain from executing incomplete installed versions* characterised by missing operational or configuration data. This ensures that operators can confidently rely on the CPS in use, possessing a code version that is not only trusted and integrity-verified, but also one that is complete without any missing

parts. This approach thereby prevents situations similar to the Airbus A400M crash, where essential calibration data was missing after take-off.

3a. Reevaluation (Range Validation) Reevaluation of sensor readings and application signals is a core functionality found in most of the specific protective shells developed. It *provides plausibility checks* on isolated sensor, actuator or application data by validating if they fall within predefined operation ranges. For example, the best temperature sensors cannot measure below -273.15°C due to physical barriers, so if a sensor reading states -300°C, it is certain that this value is erroneous.
Similarly, the CPS defines safe operation ranges for its actuators, and if an actuator report falls outside these ranges, it is likely that the actuator poses a danger to human life or property, even without interaction with other CPS components. For instance, the rotation frequency of centrifuges used in a nuclear facility operating outside the safe range, thereby destroying themselves.

3b. Cross-Validation Cross-validation of data streams traversing through the protective shell is another commonly implemented mitigation mechanism in the presented scenarios. It expands the isolated view of the reevaluation mechanism by considering data from other sensors, application signals, or actuator reports that may be dependent or correlated with each other. Domain experts define these correlations by establishing rule-sets or functional specifications, or, as described later in point 5, arise from intrinsic information within the trained behaviour model. If a data point does not adhere to these established correlations, it serves as an anomaly indicator, prompting the protective shell to investigate further.

3c. Time-sequential Analysis (Retrospective Data Evaluation) Time-sequential analysis, also known as retrospective data evaluation, widens the isolated view of the reevaluation mechanism not by considering additional data streams but by evaluating entire time series of the same data stream. This anomaly detection mechanism is used in some of the presented specific shells and offers real-time anomaly detection in certain scenarios. Common trends or variations in data series are part of the established desired behaviour for the protective shell to compare against. Sensor readings, application signals, and actuator reports are evaluated based on their previously sent data values.
If a data stream does not follow the expected trend/variation or pattern, it serves as a strong anomaly indicator. For example, sudden jumps in sensor readings are likely to indicate an anomaly because abrupt changes are uncommon in the physical world when the sampling rate is high enough.

3d. Supplementary Sensors While not universally applicable due to the additional implementation effort and its impact on the overall architecture, some specific protective shells employed supplementary sensor data as a new information stream utilised in further mitigation techniques. *Supplementary sensors are employed when protective shell decisions cannot be reliably made due to missing context information*. They provide exceptional useful data and enhance the flexibility of the protective shell. For example, in cases such as the Therac-25 or the Boeing 737 Max, the protective shells greatly benefited from supplementary sensors in efficiently safeguarding the CPS.

4a. Monitor Communications Communication and message monitoring by analysing sent packets in the CPS component network is a useful mechanism to identify primarily malicious

intent and sometimes failure. This mechanism is often employed by protective shells specifically safeguarding against security incidents. The monitoring process can trigger protective actions, such as discarding malicious messages that attempt to initiate undesired behaviour or exploit vulnerabilities in components.
The identification of abnormal packets often relies on previously trained behaviour models and patterns (will be further elaborated in chapter 6), as well as the desired behaviour specifications established by domain experts.
An example of this communication monitoring providing added security capabilities was shown in the specific protective shell for the Uconnect system, which effectively slowed down brute-force attacks and prevented the sending of arbitrary CAN- and D-Bus messages.

4b. Monitor Processes Process monitoring was implemented in various specific protective shells and provided both safety and security enhancements for the underlying CPS. It is especially useful when there exist critical tasks in the CPS that cannot be fully trusted due to software defects or security concerns.
The monitor mechanism relies on predefined functional specifications, rule-sets, or models. For instance, the specific protective shell for Toyota's ETCS employed process monitoring to detect critical but unreliable tasks that crashed under specific operation condition or due to software defects that were previously undetected. With this monitoring in place, the protective shell could detect such failures and initiate corrective actions.

4c. Monitor IAM Monitoring identity and access management processes were implemented only in the specific protective shell for Ukraine's power grid SCADA systems. Nevertheless, it provided significant security enhancements for the overall system by closely monitoring login and access attempts made by employees that formerly just got approved if username and password were correct.
IAM monitoring evaluates login requests and similar actions with a broader view, considering additional attributes such as time, IP address, and the requested resource. The monitoring mechanism relies on behaviour models trained and established through suitable machine learning techniques. By monitoring employee login patterns, the protective shell could identify anomalous behaviour if employees logged in outside office hours or from unknown locations. Additionally, it could identify abnormal access attempts, such as accessing higher than usual amounts of resources or initiating more actions than usual within the operational technology (OT) network. Upon detecting such anomalous behaviour, the protective shell could dynamically restrict access to prevent potential security breaches. This dynamic response mechanism helped to proactively mitigate security risks and safeguard the integrity of Ukraine's power grid SCADA systems.

5. Behaviour Model Training Behaviour model training and advanced anomaly detection based on the established behaviour baseline were used as mitigation mechanisms in many specific protective shells. This approach involves importing vast amounts of real-world operation data and log files from the embraced CPS into machine learning and information retrieval algorithms. These algorithms then produce a behaviour baseline that represents normal CPS operation scenarios, serving as a template for what is expected from the CPS. *Behaviour model training is often a prerequisite for other mitigation mechanisms to function effectively*. It is a versatile and almost universally applicable tool that deserves a closer look in this thesis.
This mechanism will serve as a bridge to the next chapter 6, where behaviour model training leveraging artificial intelligence algorithms will be explained in more detail, along with the presentation of advanced anomaly detection capabilities.

5.8.3 Conclusion

This section concludes the analysis of well-documented safety accidents and security incidents, with a primary focus on identifying key weaknesses and vulnerabilities within the deployed CPS. The objective has been to develop specific protective shells tailored to remediate these identified failure modes and attack vectors, followed by an evaluation of the effectiveness of each protective shell. The mitigation mechanisms employed within each developed protective shell are summarised in table 5.2, along with an assessment of the overall effectiveness in mitigating the identified vulnerabilities and ensuring safe and secure CPS operation.
Furthermore, this marks a pivotal point in this thesis where the first research question can be addressed: "Can the trustworthiness of safety and security-critical cyber-physical systems be improved by implementing a protective shell?". The evaluation of those seven case studies has clearly demonstrated that the *specific protective shells offer valuable mechanisms to address run-time issues, thus enhancing the trustworthiness of CPS during operation*. Therefore, the answer to this question is a resounding "yes".
However, the ultimate effectiveness of a protective shell once it is implemented depends on the extent to which the identified vulnerabilities fall within its realm of influence. For instance, if vulnerabilities are exploited within security incident attack paths that do not originate from the CPS, the protective shell has limited control over addressing them. Furthermore, when weaknesses are rooted in poor development practices or inadequate training of CPS operators, the protective shell is powerless to rectify such issues. Consequently, the evaluation, as depicted in table 5.2, varies significantly from "completely mitigated" to "minimally mitigated" based on the number of failure modes and attack vectors that reside within the CPS and are therefore accessible for remediation by the protective shell. Nevertheless, since the research question is formulated qualitatively, it is safe to assert that each of the seven protective shells had a positive impact on enhancing the trustworthiness of the individual safety and security-critical CPS. This chapter lays the groundwork for the subsequent chapters, especially for chapter 7 where a more detailed exploration of a generic protective shell approach is undertaken, further advancing the applicability of protective shells in safeguarding CPS.

Name	Incident / Accident	Used Mitigation Mechanisms	Protective Shell Evaluation
Boeing 737 Max MCAS	Accident	1a, 1b, 3a, 3b, 2c, 3d, 4a, 5	Mostly Mitigated
Therac-25	Accident	1a, 1b, 2b, 3a, 3b, 3d	Completely Mitigated
Stuxnet	Incident	1a, 1b, 2a, 2b, 3a, 3c, 4a, 5	Partially Mitigated
Toyota "Unintended Acceleration" ETCS	Accident	1a, 1b, 3a, 3b, 3c, 4b, 5	Partially Mitigated
Jeep Cherokee Hack	Incident	1a, 1b, 2a, 2b, 4a, 4b, 4c, 5	Completely Mitigated
Ukrainian Power Grid Attack	Incident	1a, 1b, 2a, 4a, 4c, 5	Minimally Mitigated
Airbus A400M FADEC	Accident	1a, 1b, 2a, 2b, 3a, 3b, 3c	Mostly Mitigated

Table 5.2: Overview Mitigation Mechanisms employed per chosen Accident and Incident

6 AI

The retrospective view of the previous chapter offered an in-depth exploration of the development of specific protective shells, utilising deterministic, known information from past safety accidents and security incidents. By analysing log files, operational data, and post-incident/accident reports, previously hidden vulnerabilities and weaknesses were identified, forming the foundation for mitigation mechanisms implemented in the protective shell.
Nonetheless, while this academic scenario proves valuable in comprehending protective shell architecture characteristics and identifying their commonalities, it offers limited practical guidance to CPS manufacturers seeking to enhance the trustworthiness of their existing systems. Consequently, the mitigation mechanisms alone without advanced anomaly detection, provide limited and often insufficient protection against unknown, unexpected and emerging vulnerabilities. In the modern world, large and complex systems face run-time errors and unpredictable behaviour due to various contributing factors, as exemplified by the concept of impossible results discussed in section 4.1. Consequently, it is *crucial to equip such the protective shells of complex CPS with the ability to detect and counteract hidden vulnerabilities* in the system and emergent behaviour that may threaten system safety and security (i.e. the two system quality attributes focused in this thesis).
Mitigation mechanisms, as presented in section 5.8.1, often rely heavily on advanced anomaly detection, where real CPS behaviour is compared to precalculated and trained behaviour models. Those get established through information retrieval and machine learning algorithms analysing vast amounts of operational data and log files, thus forming the behaviour baseline.
This section explains the whole concept of behaviour model training, as already teased and partly used in previous chapters. By equipping the protective shell with such models it allows the *switch from a retrospective view of vulnerabilities and weaknesses to a present view*, enabling it to enhance its embraced system's resilience against unexpected and unpredictable yet unknown behaviour, endangering safe and secure operation.
The section begins with a short introduction to anomaly detection, covering different types of anomalies and training methods. The term "*Explainable Artificial Intelligence*" (XAI) is then elaborated, and its attributes, such as accuracy, interpretability, and intrinsic explainability, are defined. Additionally, the importance of specific ML model characteristics in safety- and security-critical domains is discussed, followed by an introductory tutorial of some suitable ML models that possess the recommended characteristics. The information retrieval and training processes for these models are elaborated and illustrated how they can effectively detect anomalies.
Finally, the section highlights predictive maintenance as one of many scenarios where advanced anomaly detection, based on the presented ML models, can provide CPS with in-

sights and capabilities to monitor even the slightest deviations, thereby protecting them from far worse outcomes when left undetected.

6.1 Explainable AI (XAI) for Anomaly Detection

> When I think about anomaly detection, I immediately think of my own intuition, my Spidey-sense that something is not quite right. You know that feeling, that early warning deep inside, where your subconscious seemingly known something before you do, right? That's your internal anomaly detector at work, using rules and patterns that you might not even realise you have set up.
> *- Jeff Barr, 2023, VP and Chief Evangelist at Amazon Web Services*

How great would it be to equip safety- and security-critical systems (especially CPS) with such "Spidey-senses" to enable early detection of potential dangerous scenarios approaching or to employ some form of self-interpretation/self-awareness if they currently are operating as expected. This self-awareness is a prerequisite for initiating "self-healing" actions, such as corrective interventions of operating parameters. The machine learning-powered anomaly detection serves as the counterpart to human's natural intuition, allowing systems to identify and respond to unknown emergent behaviour that threaten safe and secure system operation. However, *merely equipping systems with mathematical intuition in the form of models that offer little insight into their functioning and decision-making raises serious concerns about their trustworthiness.* The black-box character of such machine learning models renders them unsuitable for implementation in safety- and security-critical systems, as considered in this thesis.
CPS exposing high risks to users, bystanders, or the environment, and those capable of causing costly property damage, must comply with regulations and need to earn users trust first. This is where explainable AI (XAI) comes into play, providing explanations for the model's decision-making process and ensuring that the system's actions are reflecting the constant effort to operate within the defined and desired behaviour.

6.1.1 Anomaly Detection

Anomaly detection finds widespread use in various domains, often serving a mission-critical role in risk reduction

Definition Anomaly Detection:

> Anomaly detection is the process of discovering outliers, abnormal patterns, and unusual observations that deviate from established normal behaviours and expected characteristics in a system, dataset, or environment.
> *- Cheer Simon, Deep Learning and XAI Techniques for Anomaly Detection [18]*

Detecting abnormal patterns within long-standing systems, which have accumulated extensive amounts of operational data, can be challenging due to the rarity of outliers. However, advancements in information retrieval and machine learning have shown promising results in addressing the rarity of anomalies, modelling complex high-dimensional data, and identifying novel, previously unknown anomalous classes [18]. Types of anomalies:

- Point Anomaly: An individual data point deviates significantly from expected or typical values. For instance, a sensor in a CPS may exhibit point anomaly if its reading significantly differs from typical values during normal operation.

- Collective Anomaly: A group of correlated data points, either from different sources or the same source, collectively deviate from expected values. For instance, the Therac-25 applying a high radiation dose during field-light mode.
- Contextual Anomaly: An anomaly of a single data point or a group of data points when viewed against contextual attributes such as location and time. For instance, when the Ukraine power grid attackers logged in with stolen credentials, a contextual anomaly surfaces.

While retrospective analysis of operational data after an accident or incident occurred makes the identification of anomalies simple, setting boundaries for normal and abnormal behaviour beforehand can be challenging. *Overfitting* occurs if domain experts and engineers define the desired behaviour too strictly, resulting in numerous false positive identifications of anomalies.
Conversely, if they define desired behaviour too imprecise, the model may include not only valid operational states but also some failure modes, leading to numerous false negatives where anomalies go undetected.
Balancing the model's representation of the training data how close the model should reflect them is crucial and depends on risk tolerance levels and domain-specific considerations. There is no one-size-fits-all anomaly detection solution, and thus, section 6.2 presents a variety of anomaly detection models found to be suitable for protective shell scenarios. This offers engineers implementation suggestions and templates to configure and design a protective shell that suits their specific CPS.

Anomaly Detection Training The training procedure for anomaly detection typically falls into three classes:

- Supervised: The machine learning model is trained on a labelled dataset. In the protective shell context, the label is either normal or abnormal operation and is best based on actual observation of the system. That means, if the CPS has experienced a failure mode, an incident or accident, the resulting operational data would be labelled as abnormal. Otherwise, if the CPS successfully served its purpose, the operational data would contribute to the normal behaviour category. The scarcity of abnormal data records and their imbalance compared to normal data records can make precise representation of the normal class challenging [18].
- Semi-supervised: The machine learning model is trained on a large amount of unlabelled and a small amount of labelled data records.
- Unsupervised: The machine learning model is trained on unlabelled data records. The identification of outliers solely depends on the assumption, that they are less likely to appear.

In safety- and security-critical environments, supervised learning is strongly recommended. CPS engineers and domain experts can oversee the system and identify drifts to failure better than in non-CPS scenarios. Therefore, they can aid the training process by labelling data records accordingly, contributing to more effective abnormal behaviour detection. Furthermore, in the event of safety accidents or security incidents, the ML model can be retrained using the new recorded instances of labelled abnormal operation.
The unsupervised approach is not suitable in this scenario, as domain experts must oversee the training process to enhance confidence in the resulting model.
The semi-supervised training approach may be suitable for certain cases where labelling all gathered operational data is impractical. Hence, labelled data serves as the "gold standard" due to manual mapping by experts, it requires additional work effort. If labelling vast

amounts of operation data is not feasible, the semi-supervised approach becomes a more suitable alternative than training the entire model on completely unlabelled records.
Regardless of the training approach, the main objective is to establish a *precise boundary between truly normal and abnormal behaviour*, as this determines the model's accuracy of the anomaly detection predictions.

Furthermore, it is crucial that the model's training process employed in the protective shell remains isolated from the operational phase. A diligent and systematic approach should be taken, wherein the model undergoes extensive training to a satisfactory extent, under the supervision of domain experts, followed by rigorous testing and reassessment, before being deployed into the safety and security-critical operational environment.
This *isolated training approach* for the machine learning model is particularly advisable within the context of the protective shell due to the numerous benefits it grants. Notably, this methodology provides robustness, stability, and produces more predictable outcomes, as opposed to models undergoing frequent updates during active operation. The isolation also enables extensive testing and validation, thereby ensuring to meet specifications and performance metrics. Moreover, this rigorous testing reduces the risk of unexpected behaviour during run-time, thus enhancing overall operational reliability. Lastly, the isolated approach provides data consistency within training records, serving as a protective measure against potential adversarial attacks initiated by external entities. Operational data records captured during real-time operation may significantly deviate from the preprocessed training dataset, while isolation enables essential preprocessing steps such as outlier removal, feature scaling, normalisation, and over- or undersampling.
An alternative training methodology, commonly employed in modern deep learning applications, is *continuous reinforcement learning*. This approach offers higher adaptability by continuously training the model using newly encountered operational data, allowing the model to adapt to changing conditions. Moreover, this strategy leverages real-time feedback within a feedback loop to swiftly update model behaviour based on user interactions and operational outcomes, with the objective of progressively enhancing overall model prediction accuracy over time.
However, within the context of the protective shell, it is strongly suggested to restrain from utilising continuous reinforced learning, favouring instead the well-structured sequential, isolated training process. Continuous reinforcement learning, with its frequent model updates, finds particular utility in domains necessitating high adaptability, wherein deploying immature models paves the way to mature ones. For instance, the example of Tesla's autopilot assistant training, gathering vast amounts of operational data to conduct reinforcement learning to its current model, exemplifies this approach, with the ultimate goal of achieving level 5 autonomous driving capabilities.

6.1.2 Explainable Artificial Intelligence

Definition XAI:

> XAI is an emerging research field that studies the approaches, methods, tools and frameworks to provide human-legible explanations about the inner workings of an AI system and increase confidence in model predictions with quantifiable factors.
> *qtd. in. - Cheer Simon, Deep Learning and XAI Techniques for Anomaly Detection [18]*

The primary goal of XAI is to reliably satisfy the interests and expectations of all stakeholders within the CPS's application domain. This includes regulatory requirements that CPS manufacturers must comply with by law, necessitating explanations for the model's decision-making process that then allows evaluation of regulatory compliance. Furthermore, the

manufacturer has a high interest in fully understanding its CPS, thereby improving its overall architecture by enhancing accuracy and reliability over time. Therefore, *ML model explanations should shed light on the inner workings of the protective shell's anomaly detection process*. Lastly, the CPS operator and bystanders are expecting the explainable nature of the CPS decision-making to trust the system's abilities. Social trust in a CPS is a key success factor for its acceptance. For instance, autonomous vehicles lacking human-understandable decision-making may be perceived as violating ethical guidelines, hindering the acceptance rate in the society.

Accuracy vs. Interpretability Better performing, more accurate, and efficient ML algorithms often rely on higher complexity, reducing their inherent *interpretability* for humans. Deep learning neural networks exemplify this characteristic, providing accurate predictions with low computational overhead during operation but lacking interpretability, making it challenging to consistently deliver human-understandable explanations for their actions. On the other hand, inherently interpretable models, like linear regression, provide less accurate predictions. As depicted in the figure 6.1, those two attributes are counteracting, and it relies within the engineer's choice to pick an ML model upon their requirements. The trade-off between model accuracy and interpretability must be carefully balanced to comply with regulations and fulfil functional requirements. *In safety- and security-critical scenarios, a more conservative approach is advised, favouring less complex models to reduce the likelihood of unexpected behaviour* exhibited directly by the protective shell.

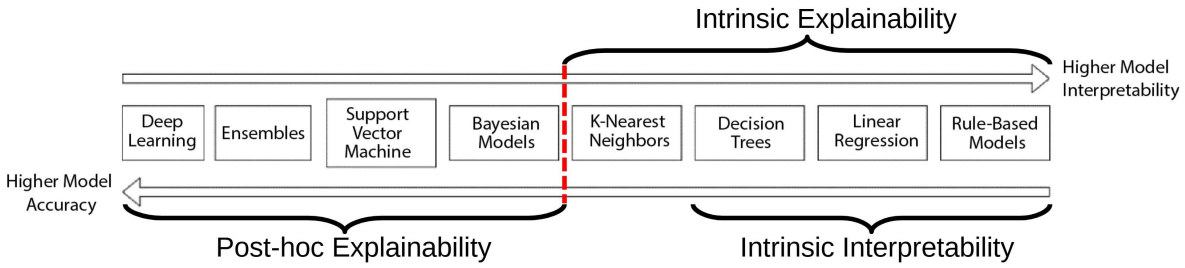


Figure 6.1: ML Model Accuracy vs. Interpretability [18]

The main distinction between explainability and interpretability lies in the scope of their contributions. Explainability provides reasoning for the model's decisions, while interpretability measures the extent to which the model's overall behaviour is understood. That means, that explainability shows how the model behaves in human understandable terms. In contrast, interpretability assesses whether the given explanations of how the model behaves make sense in real-world practice [18].
In the safety- and security-critical CPS context, interpretability is welcome aiding in obtaining a deep understanding of the model, while explainability is essentially required. However, more complex, multidimensional models may lack interpretability and intrinsic explainability while providing higher efficiency and accuracy in predictions. Therefore, active and emerging research focuses on providing post-hoc explainability for such complex models.

Intrinsic Explainability vs. Post-hoc Explainability Machine learning models exhibit intrinsic explainability by limiting their overall complexity and dimensionality during the initial design phase. In contrast, machine learning models with high complexity and high-dimensional datasets lack intrinsic explainability and rely on post-hoc explainability, employing algorithms to calculate explanations and analyse the model's behaviour after training. In the safety- and security-critical CPS domain, it is essential to *avoid post-hoc explainability*, as the goal is not to find a mathematically suitable explanation of what happened but to identify the root cause

of the protective shell's actions.
A significant advancement in the domain of AI explainability took place in June 2023, when the European Union Parliament ratified their Artificial Intelligence Act (EU AI Act). This legislative milestone marks the start of lawmaker negotiations aimed at finalising the new regulatory framework. This legislative act underscores the necessity of explainability in AI systems, stating: "Members seek to strengthen citizens' rights to file complaints about AI systems and receive explanations of decisions based on high-risk AI systems that significantly impact their rights." [34]. Foreseeably, this legislative direction will serve as a guiding force for establishing the initial legal norms on a national level across the European Union.
Intrinsic explainable models, as shown in figure 6.1, are intentionally designed and constructed with simple structures to be self-explanatory. Conversely, ML models requiring additional models or post-processing to provide explanations are categorised having post-hoc explainability [18].
In models with intrinsic explainability, the influence of a feature (independent variable) on its target (dependent variable) can be determined by examining the weights and biases for linear and logistic regression. Therefore, explanations for specific predictions can be obtained by analysing the weight matrix and attention scores in each model.
Conversely, post-hoc explainable models lack such linear or logistic coefficients that can be directly visualised as explanations. Consequently, for every prediction that requires an explanation, the model must be evaluated using an explainability algorithm, approximating the model's behaviour as best as possible. This approximation legitimately raises concerns about the explanations' faithfulness and trustworthiness, as it is not a direct representation of the model's behaviour. Post-hoc explainability algorithms often focus on feature importance, assessing how model accuracy deviates after permuting the values of a specific feature to obtain prediction scores [18]. For instance, deep neural networks lack interpretability and intrinsic explainability for humans, relying on post-hoc explainability methods to shed light on the hidden layers' inner workings. One such tool is the activation maximisation framework, which provides insights into preferred inputs for specific layers of neurons and their activation.
Figure 6.2 demonstrates the search for an explanation for the output of a neural network using a gradient-based backpropagation algorithm for feature attribution. This algorithm assigns importance scores at the final output layer and calculates local activation gradients across each intermediate layer until it reaches the input layer, thereby producing a viable explanation for the output [18].

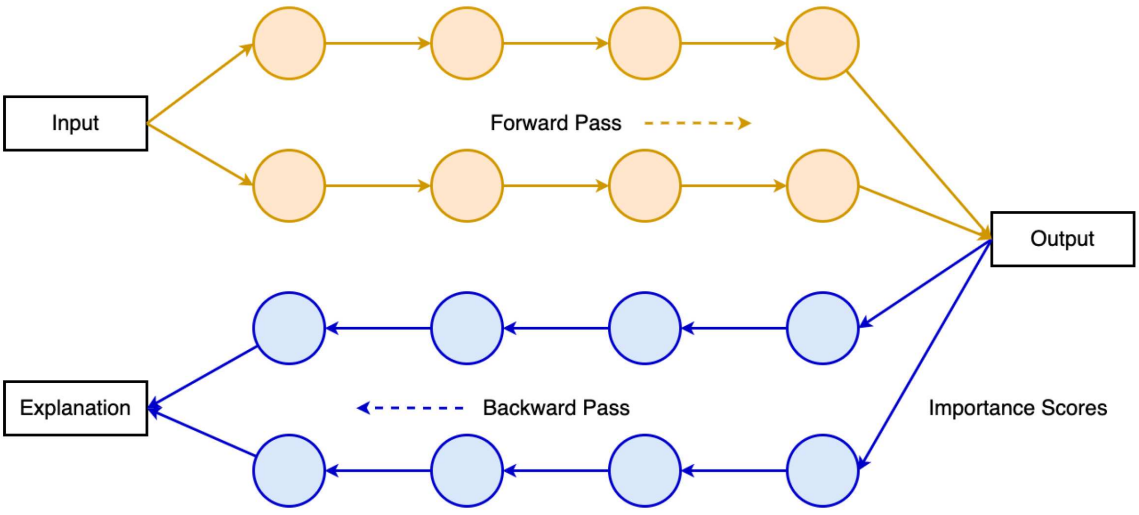


Figure 6.2: Post-hoc Explainability - Neuronal Network Backpropagation [18]

Ignoring explainability by solely focusing on complex, high-dimensional models that deliver consistently more accurate predictions is not a valid approach. During design time, engineers and domain experts may not be able to consistently develop a model that accurately and adequately reflects real-world scenarios. As a result, the model's predictions may be highly accurate based on the training data, but the model may deviate substantially from the real-world problem. Reevaluating the suitability of the established model in addressing real-world problems is crucial, especially in safety- and security-critical CPS domains. Regulatory compliance cannot be achieved, and users won't rely on an AI-supported CPS that is not considered trustworthy.

In summary, in the CPS context, *an intrinsic explainable model with adequate accuracy is always preferable to a complex model with high accuracy* [18]. Therefore, the following section presents a selection of intrinsic explainable ML models that extract business-critical insights and achieve transparency, ultimately establishing trust in the system. Some of these models are inherently interpretable, while others lack this attribute in exchange for more complex ML techniques.

6.2 Intrinsic Explainable ML Models

Traditional rule-based anomaly detection methods, manually crafted by domain experts, have historically offered inherent interpretability. However, these methods face limitations in their scalability when dealing with high-dimensional data and are challenging to maintain [18]. As a result, they are best suited for capturing fundamental system characteristics that are expected to remain stable. Their rigid nature allows them to define broad boundaries based on functional specifications and policies, but if applied to more dynamic and flexible limits, they can hinder system development.
It should be noted that this section provides a concise introduction to suitable ML models for protective shells. Therefore, certain aspects not directly relevant to this thesis have been omitted in order to keep the scope of content about AI to an acceptable level.

6.2.1 Linear Regression

Concept Linear regression is a fundamental machine learning technique employed to model the relationship between a dependent variable (target) and one or more independent variables (features). This technique assumes a linear model as its foundational learning framework. In the linear model, the target $\hat{y} = f_{w,b}$ is expressed as a linear combination of input features $X_1, ..., X_n$ [35].
One of the key advantages of linear regression is its inherent interpretability, coupled with having an analytical solution that enables learning of the optimal linear model [35]. The goal is to find a linear equation that best represents the relationship between these features. The linear regression equation, along with a bias term, has the following form:

$$f_{w,b}(x) = \vec{w} \cdot x + \vec{b}$$

Where:
$f_{w,b}$ - represents the target values the model aims to predict
x - denotes the independent features
$\vec{w}$ - represents the weights
$\vec{b}$ - represents the biases

Training/ Information Retrieval Training a linear regression model necessitates a labeled dataset consisting of input-output pairs, thereby employing supervised training. Throughout the training process, the optimal values for $\vec{w}$ and $\vec{b}$ need to be determined, minimising the discrepancy between predicted values and actual values within the training dataset. Typically, this is achieved using an optimisation algorithm such as gradient descent, iteratively adjusting the weights until convergence.

Detection Process Linear regression can be applied in two variants for anomaly detection during run-time. The first method involves identifying anomalies within a sequentially ordered stream of data points. The linear regression model is trained to learn data trends and variations in individual sensor readings, actuator reports, and application signals. By assessing recent values and extrapolating them into the future while considering learned common trends/variations from operational data, the protective shell can compute the anticipated value likely to be observed next in the data stream. As illustrated in figure 6.2.1, this regression task calculates an expected value for each sensor, actuator, and application signal in the forthcoming sampling frequency iteration.

$$\left\{\begin{array}{c} Sensor1_{t-2} \\ Sensor2_{t-2} \\ \dots \\ Actuator1_{t-2} \\ Actuator1_{t-2} \\ .. \\ ApplicationSignal1_{t-2} \\ ApplicationSignal2_{t-2} \end{array}\right\}, \left\{\begin{array}{c} Sensor1_{t-1} \\ Sensor2_{t-1} \\ \dots \\ Actuator1_{t-1} \\ Actuator1_{t-1} \\ .. \\ ApplicationSignal1_{t-1} \\ ApplicationSignal2_{t-1} \end{array}\right\}, \left\{\begin{array}{c} Sensor1_{t} \\ Sensor2_{t} \\ \dots \\ Actuator1_{t} \\ Actuator1_{t} \\ .. \\ ApplicationSignal1_{t} \\ ApplicationSignal2_{t} \end{array}\right\} : \left\{\begin{array}{c} Sensor1_{t+1} \\ Sensor2_{t+1} \\ \dots \\ Actuator1_{t+1} \\ Actuator1_{t+1} \\ .. \\ ApplicationSignal1_{t+1} \\ ApplicationSignal2_{t+1} \end{array}\right\}$$

Additionally, the model learns the standard deviation for each data stream, which is utilised to set an anomaly detection threshold. This threshold, defined by lower and upper bounds, enables anomaly detection for each data stream validating the next value against this range, as depicted in figure 6.3.

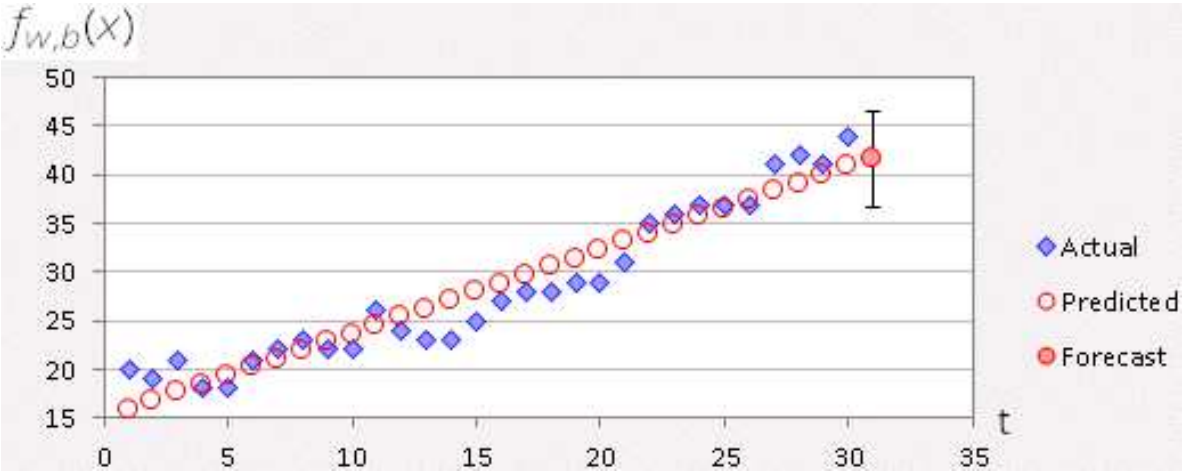


Figure 6.3: Linear Regression - Threshold Anomaly Detection [36]

The second approach employing linear regression for anomaly detection operates at a system-wide level. Leveraging the linear classification strategy, the model predicts whether the entire CPS is functioning normally or experiencing a failure mode. The model is trained to assign a positive $f_{w,b}$ value to all datasets indicative of "operating as intended" and a negative $f_{w,b}$ value to those indicating "failure modes". To achieve this, the model learns to span an optimal straight-line boundary that separates failure mode points situated below the line from correct operation points located above the line, as shown in figure 6.4.
Once this boundary is defined (employing appropriate mathematical techniques to project multidimensional points onto a single layer), the linear classification can efficiently determine the current operational state, identifying potential failure modes or correct operation.

It is important to acknowledge that linear regression alone might not be sufficient for complex anomaly detection tasks, especially when variables exhibit non-linear relationships. More advanced techniques such as support vector machines or deep neural network learning might be necessary to handle complex datasets with high-dimensionality and non-linearity. Nevertheless, the simplicity of linear regression renders it immediately interpretable and infuses trust in its predictions.

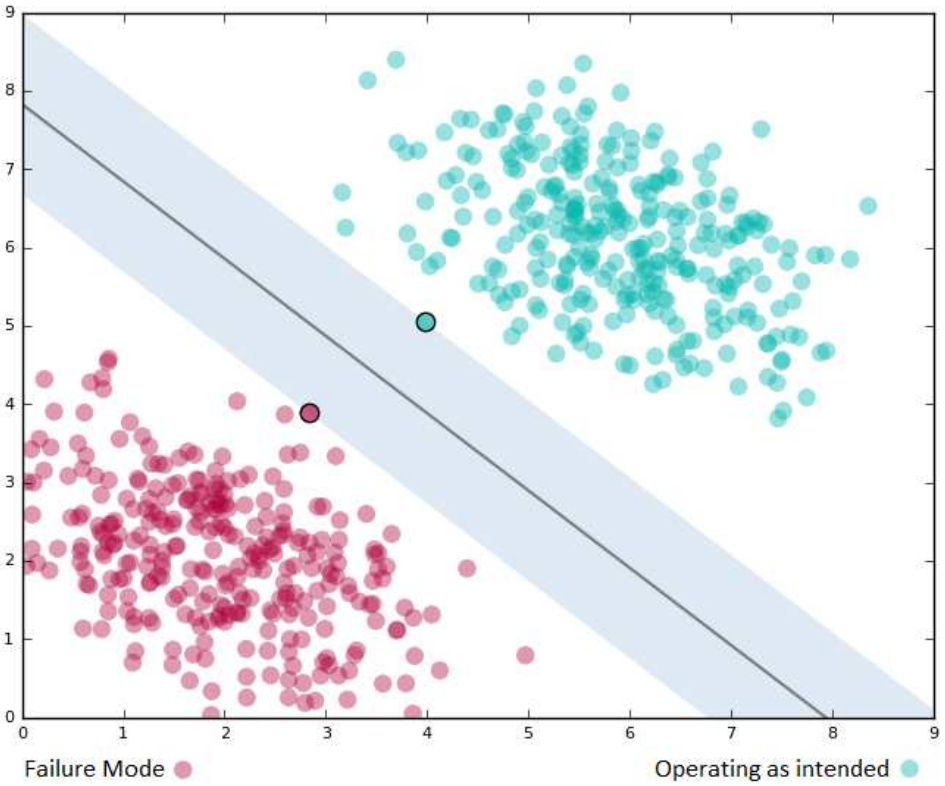


Figure 6.4: Linear Regression - Classification [37]

6.2.2 Decision Trees

Concept Decision trees are robust machine learning models utilised for both classification and regression tasks, similar to linear regression 6.2.1. In the context of anomaly detection, classification is employed to assess whether the CPS is operating according to its design or if it is likely in a failure mode. Furthermore, regression tasks are used to predict forthcoming values for the individual data streams that the protective shell is expecting next [35].
A decision tree models a decision-making process in a tree-like structure, often realised as binary trees. Internal nodes represent decisions based on specific features or attributes, while leaf nodes correspond to class labels (in classification) or predicted values (in regression tasks).
The tree is built through a process of recursively partitioning the data based on the feature that provides the best separation of classes or the best prediction of values.

Training/ Information Retrieval Two measures play a crucial role in decision tree training: entropy and the information gain.
Entropy is a measure of impurity or disorder in a set of data. In the context of decision trees, it represents the uncertainty of class labels within a subset of data. A lower entropy indicates that the subset is more homogeneous with respect to the target variable.
Information gain is a metric used to decide which feature to split on. It quantifies the reduction in entropy achieved by partitioning the data based on a particular feature. A higher information gain implies that the feature separates the data into more distinct classes, thereby improving the ability to make decisions.

The training process starts at the root of the tree with the entire dataset, as depicted in figure 6.5 for the classification use case where X_0 and X_1 are features. The feature providing

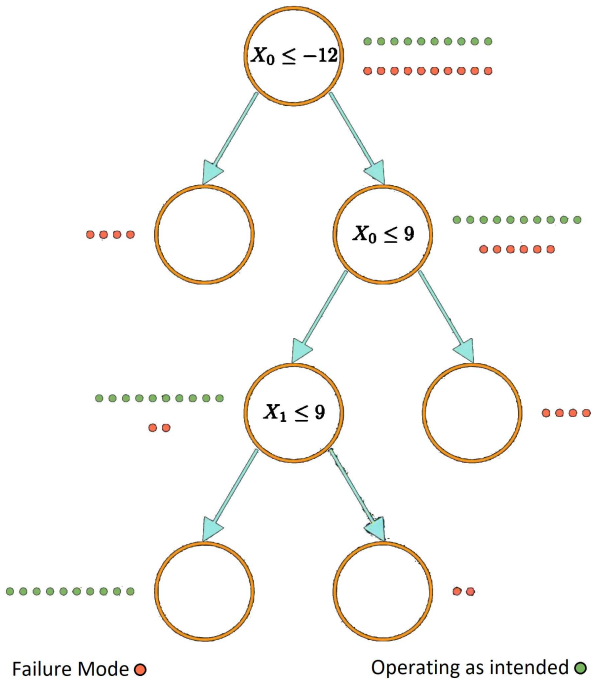


Figure 6.5: Decision Tree - Classification Training [38]

the highest information gain is selected to partition the data into subsets based on feature values, building the first internal node. This procedure gets repeated for each subset until a stopping criterion is met (e.g., maximum depth, minimum samples per leaf, or no further information gain). Upon subset purity (all instances belong to the same class) or meeting the stopping criterion, a leaf node is generated with the class label corresponding to the majority class.
To mitigate overfitting, pruning can be applied to remove branches that do not notably enhance the model's generalisation.
In summary, the training process of a decision tree involves recursively selecting the best features to split the data based on information gain, calculating entropy to measure impurity, and creating nodes and branches to form a tree-like structure. This process continues until certain stopping criteria are met, or the tree becomes pure.

Detection Process There are various approaches in leveraging decision trees for anomaly detection. A very promising approach is the concept of isolation forests. As anomalies are defined by their deviation from the majority of data records, they tend to be isolated in fewer splits, thus residing closer to the root of the tree. As shown in figure 6.6 the outliers can be isolated from the inlier area often by few splits, establishing a binary decision boundary.

Consequently, anomalies tend to exhibit shorter paths from root to their leaf due to their distinctiveness. Depth-based methods utilise the leaf node's depth as an anomaly score. Figure 6.7 demonstrates that data points requiring short path lengths for isolation often correspond to anomalies, while long paths suggest expected operational states in the CPS scenario.
In conclusion, decision tree models surpass linear regression in complexity while remaining inherently interpretable and can learn non-linear relationships in the underlying data. Despite the algorithm's greedy nature leading to a local optimal solution but can't guarantee a global optima, decision trees are efficient in yielding solutions within a reasonable timeframe. Nonetheless, they are prone to overfitting, leading to false positive anomaly de-

tection. Therefore, processes like pruning are essential to mitigate overfitting tendencies.

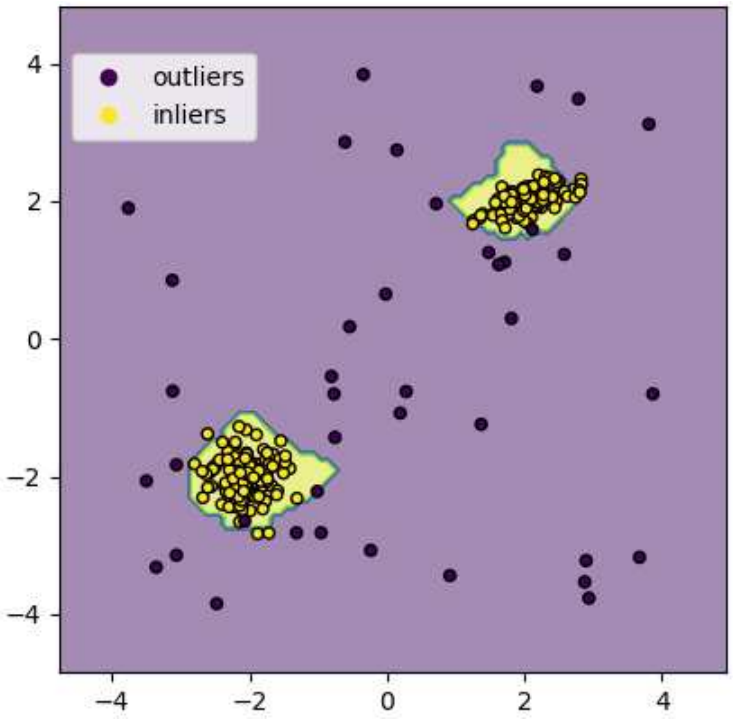


Figure 6.6: Decision Tree - Isolation Forest Binary [39]

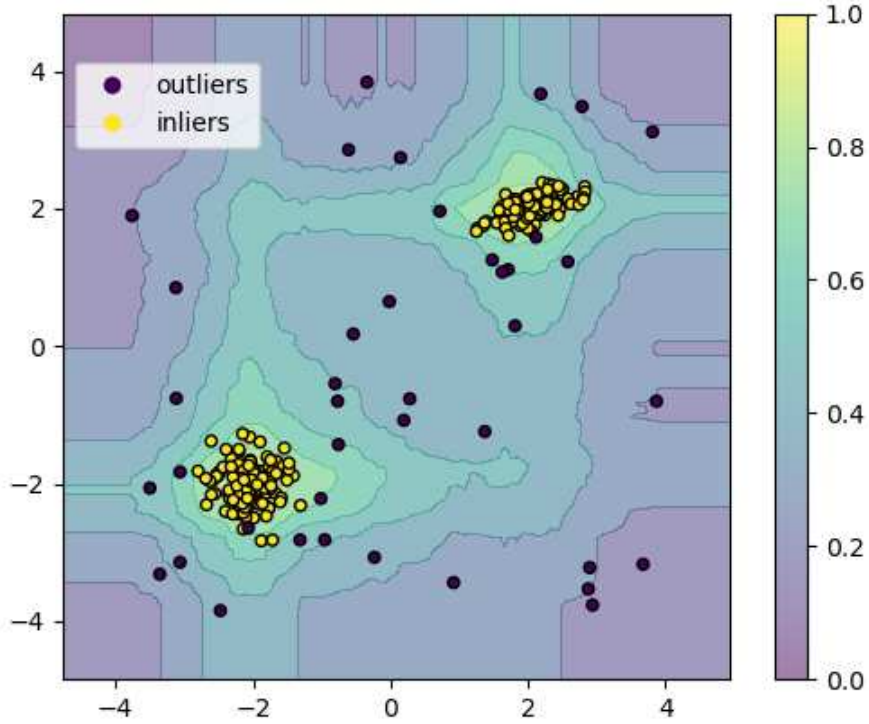


Figure 6.7: Decision Tree - Isolation Forest Depth Based [40]

6.2.3 K-Nearest Neighbours

Concept The dominant focus of numerous existing machine learning methods centres around point anomalies. However, the emergence of complex interconnected CPS, generating multidimensional data streams, pose a new opportunity for the domain of multimodal anomaly detection. Instances of multidimensional anomalies manifest when one or more dimensions deviate from the anticipated range of values, or when multiple dimensions cease to exhibit long-standing correlations [18]. Addressing this scenario, the k-nearest neighbours (k-NN) model stands as a suitable solution to detect such multidimensional anomalies, inherently possessing higher dimensionality than the previously presented models. The k-NN model is a fundamental technique within in the machine learning domain, serving various applications, including the use case of anomaly detection.

At its core, the k-NN algorithm is rooted in non-parametric and instance-based learning, operating upon the principle of similarity. Consequently, k-NN has no fixed number of parameters and instead engages in a comparative assessment between new data records and the dataset encountered during the training phase. The dimensionality of the k-NN model corresponds directly with the number of data streams traversing through the protective shell. Therefore, for every component transmitting or receiving data within the CPS, a distinct feature axis is created. While the typical k-NN model relies on labelled datasets, adaptations allow its operation with unsupervised learning paradigms.
The k-NN model proves particularly useful for identifying complex and subtle patterns in data that may not be captured by conventional parametric models such as linear regression.

Training/ Information Retrieval K-NN is not typically ”trained“ in the same way as traditional machine learning models, as it is a ”lazy learner“. Instead, the feature vectors of the training data get stored in memory, so that the algorithm can compute distances and find neighbours efficiently during the prediction phase. However, for vast amounts of training data, k-NN can leverage time-memory trade off by not saving all individual data records as points in the multidimensional space, but calculating classification spheres instead. As shown in figure 6.8, the sphere got calculated by approximating the placement area of data record points labelled as operating as intended.
The resulting model may have intrinsic knowledge about the CPS on a very detailed level, that is infeasible for domain experts to extract manually.

Detection Process The initial step involves defining the parameter ’k’, representing the count of nearest neighbours considered in decision-making. This parameter should remain static once defined, as its alteration could drastically change the model’s behaviour.
Subsequently, the newly recorded data point is located within the space, and distances to the training data points are computed. Common distance metrics include Euclidean distance, Manhattan distance, and most important, cosine similarity, which gains preference in models featuring high dimensionality because of its performance advantage.
Following, the ’k’ data points having the shortest distances to the new point get identified. Classification is accomplished through majority voting among k-nearest labels. Alternatively, a distance-based score is computed, unveiling the probability of the new point being an anomaly. Anomalies are often associated with data points that have high average distances to their neighbours, attributed to their rare occurrence.
The utilisation of precalculated spheres within the multidimensional space avoids the need for distance calculations after the sphere training, thereby enhancing operational efficiency.

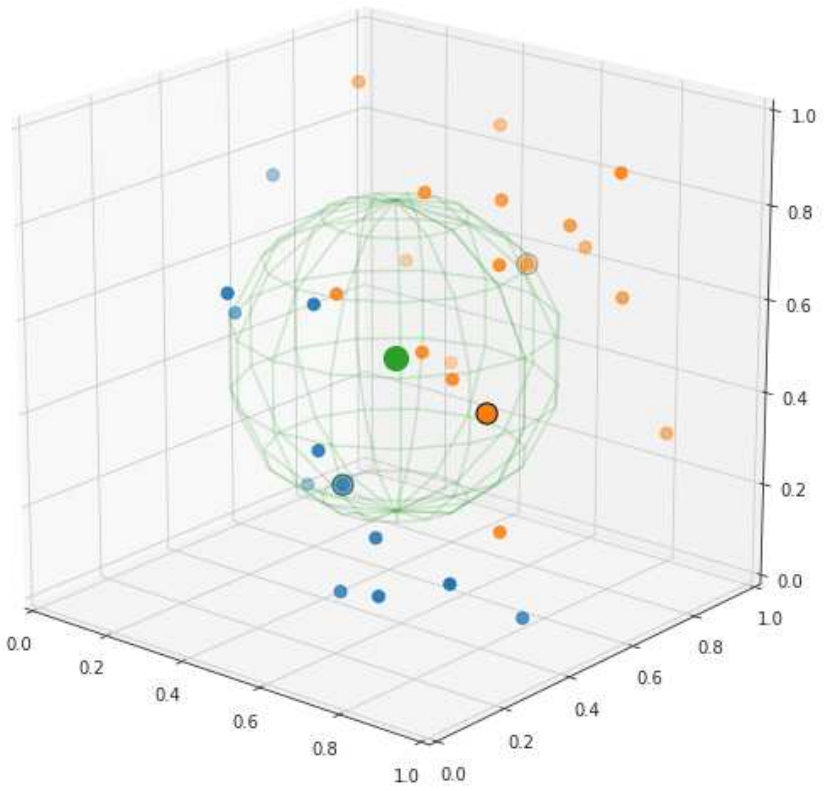


Figure 6.8: K-nearest Neighbours - Sphere Classification [41]

While k-NN provides intrinsic explainability and greater versatility in comparison to previously introduced algorithms, it is not inherently interpretable and does raise substantial performance concerns. In simple scenarios characterised by low-complexity CPS, k-NN can yield significant benefits. However, this thesis concentrates on complex CPS instances demanding high-dimensional k-NN models to accurately depict their behaviour. In high-dimensional spaces, the "curse of dimensionality" can pose tremendous negative effects on k-NN's performance. Distance-based methodologies might encounter challenges in locating meaningful neighbours within such spaces.
Furthermore, the selection of 'k' significantly influences the algorithm's efficiency. A small 'k' could lead to overfitting, whereas an overly large 'k' might cause the algorithm to overlook local patterns. For scenarios targeting over 20 features per instance (corresponding to more than 20 data streams and, hence, a dimensionality surpassing 20), k-NN proves unsuitable. Nevertheless, in scenarios involving up to 20 features and extensive training data, k-NN emerges as a highly viable choice, providing the advantage of preserving training data during training without information loss due to its non-training nature [35].

6.3 Example Use Case - Predictive Maintenance

Predictive maintenance is a crucial application of machine learning in the field of industrial operations. Real-time data produced by interconnected devices presents enormous opportunities for predictive analytics in structural health checks [18]. Through such advanced analytics and the use of machine learning techniques to predict when equipment or machinery is likely to fail, timely maintenance interventions are made possible to prevent unplanned downtime and optimise maintenance schedules. Anomaly detection plays a central role in predictive maintenance, as it helps identify deviations from normal operating conditions that might indicate impending failures or issues.
Inadequate machine maintenance is the primary cause of unplanned downtime in manufacturing. Improving equipment availability and performance is critical in avoiding unnecessary maintenance costs and increasing productivity in industrial workloads, thereby ensuring business continuity [18].
The three machine learning algorithms presented previously — linear regression, decision trees, and k-nearest neighbours — can be employed to facilitate anomaly detection in predictive maintenance scenarios.

1. Linear Regression: In the context of anomaly detection for predictive maintenance, linear regression can be utilised to establish a baseline of normal behaviour for a machine or system. By training a linear regression model on historical sensor data, the algorithm learns the expected patterns of behaviour. Deviations from the predicted values can then be flagged as anomalies like higher electricity consumption's or when an operational hour limit is reached, potentially indicating an impending failure if left ignored.

2. Decision Trees: In predictive maintenance, decision trees are useful for creating a hierarchical structure of rules based on historical data patterns. Anomalies can be identified by traversing the decision tree and observing the path that a new data point takes. If the path is uncommon or diverges significantly from the established patterns, it suggests the presence of an anomaly. Decision trees are particularly effective for capturing complex interactions among multiple variables like temperature fluctuations or pressure changes.

3. K-Nearest Neighbours: In the context of predictive maintenance, k-NN can be employed to find similar historical instances of normal behaviour. If a new data point is significantly distant from its neighbours, it indicates an anomaly. k-NN is especially useful when the decision boundary between normal and anomalous instances is irregular or nonlinear, like vibration levels and performance deviations.

Predictive maintenance is a vital application of machine learning in industrial contexts, allowing organisations to reduce downtime, enhance operational efficiency, and extend the lifespan of machinery. Anomaly detection using algorithms like linear regression, decision trees, and K-nearest neighbours empowers businesses to proactively identify and address potential issues, thereby maintaining the health and reliability of their equipment. By leveraging these techniques, *industries can transition from reactive maintenance practices to data-driven, proactive maintenance strategies*.

Equipped with the presented ML models used for advanced anomaly detection outlined in this chapter 6, the summarised mitigation mechanisms collected in chapter 5 as well as the goals, expectations and state-of-the-art from chapter 3 and 4, the subsequent chapter provides insights how a generic protective shell architecture can look like, universally applicable and dynamically adoptable for a wide range of application domains.

7 Generic Protective Shell

The necessity for a generic protective shell that can be universally applied across multiple application domains arises from the previously discussed and well-reasoned challenges faced by modern IT systems. These challenges are repeated here to underscore and recall the core issues:

1. The conventional development process, despite its sophistication, cannot identify and mitigate all vulnerabilities in the system's design-time architecture before deployment despite typically involving risk management, vulnerability elimination, system model verification, testing, and various other techniques.

2. The high complexity of modern systems is known for generating emergent properties, which can include negative emergent behaviour and vulnerabilities.

Therefore, the primary objective of this thesis is to enhance the trustworthiness of such complex systems during their operation by introducing an additional layer of defence – run-time monitoring in the form of a protective shell. In situations where complex systems exhibit emergent negative behaviour, the protective shell is designed to detect and analyse this abnormal behaviour. Subsequently, it aims to counteract impending safety accidents or security incidents by mitigating the potential damage they may cause.
This chapter introduces the concept of a generic protective shell with the objective of *achieving universal implementability across various application domains*. Establishing a dynamic protective shell framework not only enables its adaptability through blueprints and templates but also aligns with broader engineering principles, serving as a *prerequisite* for widespread industry acceptance.
The groundwork for this thesis involved an examination of the key actors, current technological landscape, opportunities, and challenges within the CPS environment. It aimed to uncover the functional requirements sought by organisations and individuals in this domain, thereby absolving the design science research relevance cycle to leverage opportunities and find solutions to exposed problems. Among the identified challenges lies the research problem addressed in this thesis, *focusing on emerging safety accidents and security incidents in complex CPS scenarios*.
Furthermore, this chapter offers a summary of pivotal methodologies, frameworks and principles drawn from the scientific knowledge base. These methodologies either partially address the research problem or provide useful functionality to be refined, adopted, and integrated into the proposed solution. Notably, it explores the integration of the AI guardian angel bots suggestion [1], the run-time monitoring principle, and various machine learning models. These elements collectively establish the foundation and provide a robust basis to build the *artefact* of the generic protective shell.

The design cycle, which already started in chapter 5, involved the development and evaluation of specific protective shells for CPS applications across various industries. This iterative process unveiled similarities, laying the groundwork for the *core of the generic protective shell framework*. Chapter 6, contributing to the rigor cycle, delves deeper into additional foundations and methodologies from the machine learning and anomaly detection domains. This enriches the design and architecture principles of the protective shell by providing guidance on important ML model characteristics and training approaches.

In this chapter, the focus lies on the core architecture of the generic protective shell framework. Firstly, the *MAPE-K control loop* is introduced and tailored to the context of the protective shell. Its suitability for the protective shell concept is elucidated.
Additionally, a comprehensive list of protective shell detection capabilities is presented, encompassing a wide range of advanced anomaly detection mechanisms, machine learning classifications, and simpler static rule-based approaches for identifying abnormal behaviour. Likewise, a list of protective shell intervention capabilities is introduced, addressing various potential outcomes of intervention decisions. These capabilities span passive alerting and active corrective/protective actions, whenever possible in real-time.
Both lists contribute to the protective shell's *plugin catalogue*, offering a wide variety of modules that can be directly implemented if they provide functionality suitable for monitoring or safeguarding the specific embraced CPS. Furthermore, an automated process for analysing and interpreting abnormal behaviour is outlined to enable purposeful and aimed protective actions.
To conclude the generic protective shell architecture, the identified *design and architecture principles* are enumerated. These principles are intended to guide engineers in the successful implementation of protective shells tailored to their specific CPS applications. Subsequently, required adaptations to the overall CPS development process necessitated by the introduction of the protective shell are discussed.
Finally, an evaluation of the generic protective shell framework is conducted, focusing on *residual risks, effectiveness, the potential introduction of new failure modes, and its impact on the attack surface*. This chapter serves to enrich the scientific knowledge base with the created artefact by offering a viable solution to the research problem, which organisations operating in the CPS application domain can readily adopt.

7.1 Architecture

The primary objective of the protective shell is to address undetected vulnerabilities stemming from the development process and emergent negative properties within the CPS's runtime environment. This section outlines a sound architecture equipped with the necessary capabilities to meet the functional requirements of protecting the system during run-time.
Firstly, the well-suited reference architecture, MAPE-K, gets introduced, followed by an in-depth analysis of each phase within this architecture.
Subsequently, two lists of "plugin/modules" get presented to provide the flexibility required to tailor the protective shell to individual CPS instances and their specific use cases. The first list covers the various anomaly detection and abnormal behaviour identification techniques that can be incorporated, while the second list encompasses the diverse intervention actions available to the protective shell.
This section concludes by consolidating all the design and architecture principles identified throughout the preceding chapters, including any novel insights discovered during the architecture's definition.

7.1.1 MAPE-K

A quite suitable *reference architecture* for the protective shell to follow is the MAPE-K model, which employs a feedback loop to continuously monitor and validate the current system behaviour. When abnormal behaviour is detected, it plans and executes adaptions to restore the system to a desired operational state.
The acronym MAPE-K represents the core phases constituting this feedback loop, standing for **M**onitor, **A**nalyse, **P**lan and **E**xecute, with each phase of this feedback loop relying on an underlying **K**nowledge base. This reference architecture is especially useful for *self-adaptive* (including self-healing) and autonomous systems within domains like CPS [42]. Self-adaptive systems, like CPS, are a class of systems that have the ability to autonomously modify their behaviour or properties in response to changes in their environment, operating conditions, or internal state to be resilient and flexible. These systems are designed to achieve and maintain certain goals or objectives while dynamically adjusting to varying circumstances. In this thesis, the MAPE-K architecture gets adopted and used for the protective shell, safeguarding the embraced CPS against run-time failures and threats.
Given that the CPS itself operates in a similar feedback loop, in this thesis the protective shell's MAPE-K loop gets referred to as the "*control loop*". Both loops are interdependent and cannot operate in parallel; they must be sequentially processed for each iterative cycle. The feedback loop awaits data streams, including sensor readings, application signals, and actuator reports, which pass through the protective shell during the control loop. Conversely, the control loop awaits new data streams sent out or requested by the CPS feedback loop. This introduces concerns regarding latency, which can lead to system instability, partial paralysis, and delayed responses. This issue will be addressed in detail during in-depth analysis of relevant MAPE-K phases and subsequently addressed through the specification of especially tailored design principles. The entire MAPE-K control loop is managed by the protective shell's application software, with each phase continuously consulting the knowledge base containing, but not limited to, run-time models, functional specifications, policies, rule-sets, machine learning models, and anomaly detection capabilities.
A concise overview of the refined MAPE-K reference architecture is illustrated in the following figure 7.1, while a more detailed depiction can be found in the appendix of this thesis.

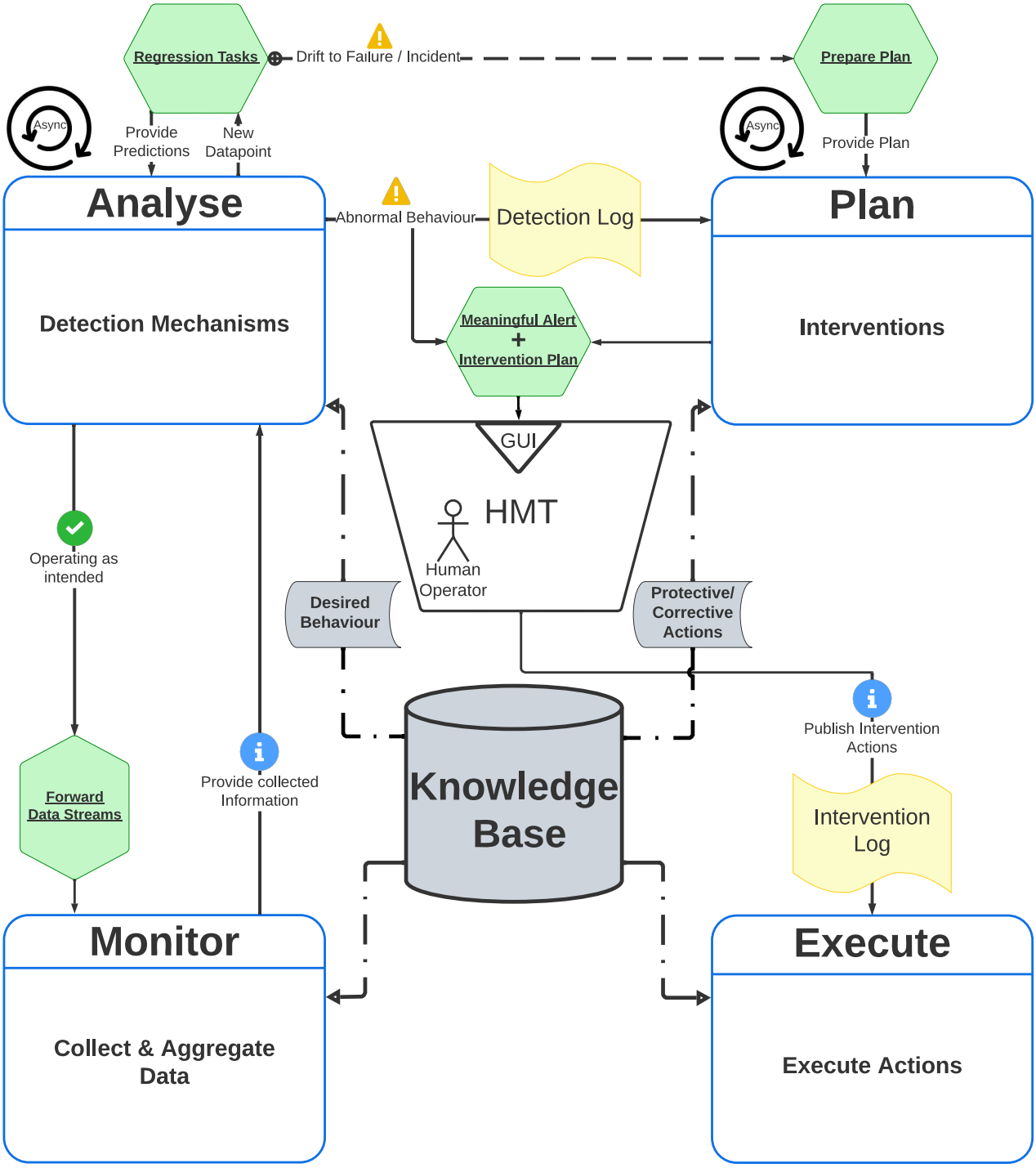


Figure 7.1: Architecture - MAPE-K Protective Shell Overview

Monitor

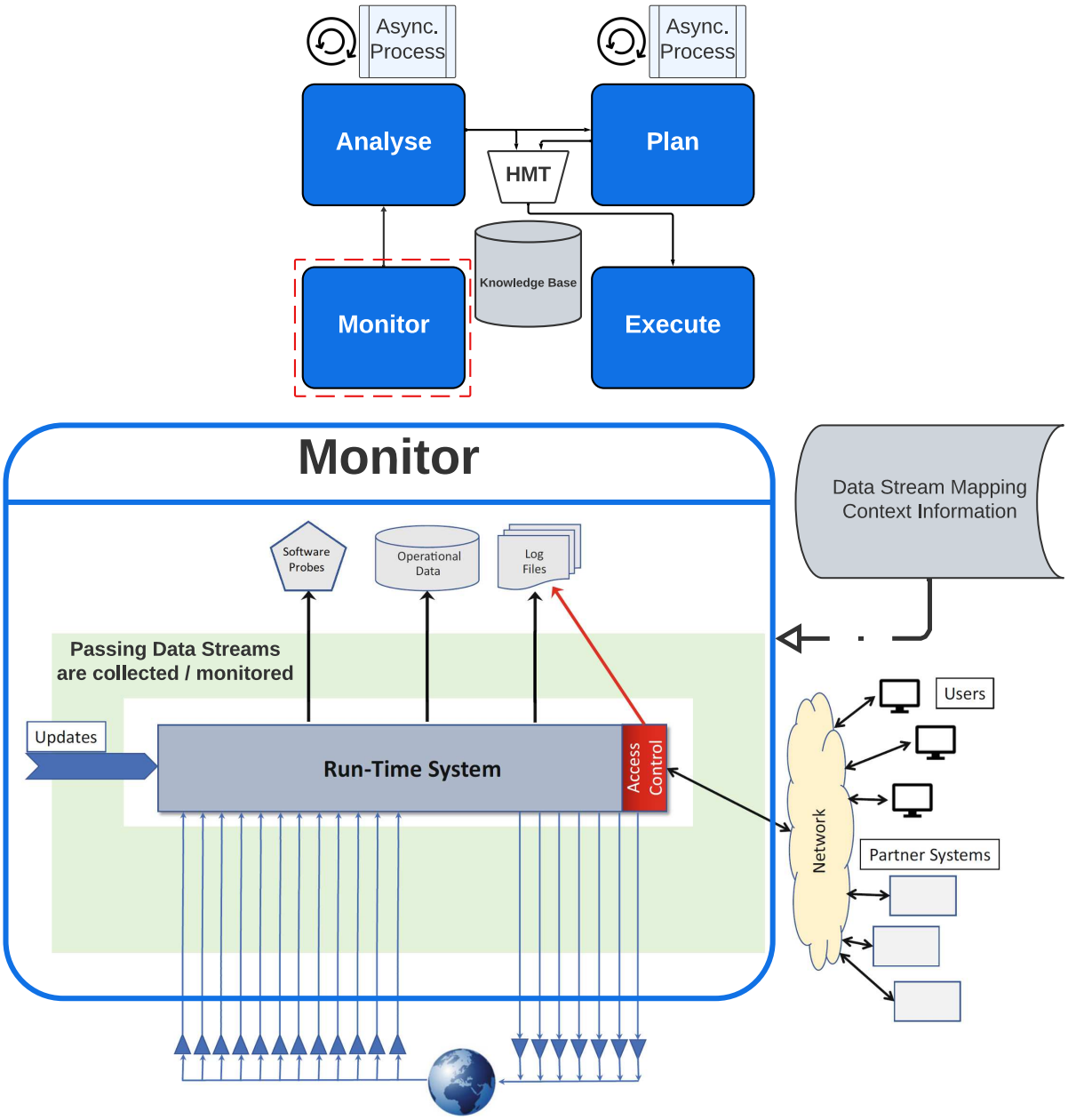


Figure 7.2: Architecture - MAPE-K Monitor Phase [2]

In the monitor phase of the control loop, the protective shell not only collects sensor readings, similar to the CPS's feedback loop but also gathers application signals, actuator reports, network access information, update requests, and additional context data sources that are indirectly linked to the CPS. The primary objective of the monitoring phase is to gather data from the self-adaptive CPS and its operating environment [42], with all information sources predefined and mapped in the underlying knowledge base that the monitor phase consults. Sensors provide the protective shell with environmental information, equivalent to what the embraced CPS requires for its operation. *Software probes*, on the other hand, offer insights into the resource usage, response times, and currently executing tasks of CPS applications. This data is collected, persisted, and utilised in run-time models to guide subsequent analysis of abnormal behaviour, anomaly detection, and decisions regarding protective and cor-

rective actions [42].
To optimise resource utilisation and ensure efficient data management, the monitor phase frequently undertakes preprocessing of the quite voluminous data streams. This *preprocessing* may encompass *data aggregation*, wherein information from diverse sources is consolidated before onward transmission to the analysis phase. By employing such measures, the monitor phase strives to alleviate storage constraints, optimise bandwidth utilisation, and reduce the computational overhead associated with handling copious data streams.

Analyse

Within the analyse phase of the control loop, the protective shell predicts the need to intervene in CPS-initiated actuator commands or other data streams linked to the CPS by analysing the information received from the monitor phase. It assesses whether adaptations are required to enhance the CPS's run-time safety or security. *This assessment hinges on whether the CPS is currently operating as intended, as defined by its desired behaviour, if it is gradually deviating from a safe and secure operational state into instability, or if it currently exhibits abnormal behaviour*.
The knowledge base underpins the analyse phase, providing the necessary information to enable anomaly detection, operating state classification, cross- and range-validation, checking correlations, as well as manual established functional specifications, rule-sets, policies, models, and patterns. These resources converge harmoniously during each control loop iteration to evaluate the current CPS behaviour and subsequently compare it against the desired behaviour.
During analysis, the protective shell predicts whether the CPS can achieve its defined goals within safety and security constraints while adhering to agreed-upon quality of service specifications. This fully automated process must enable rapid responses to environmental changes, emergent situations [42], threats through malicious intent, or faults resulting from hardware or software failures. However, the validation of CPS behaviour during analysis by the protective shell itself introduces *latency* to data streams, such as sensor readings reaching the CPS, and increases response times for actuators following CPS instructions. *Consequently, the responsiveness of the self-adapting CPS decreases, and its reaction times increase, which can lead to system instability*.
This latency is particularly problematic in CPS scenarios with high sampling frequencies and real-time requirements, such as autopilots, where response times are critical for preventing accidents. Therefore, it is crucial to optimise the analysis phase's efficiency, reducing the latency factor to a minimum. One effective strategy is to leverage regression tasks from trained machine learning models or established data trends to project previous and current values into the future, thereby predicting the next expected values. This *asynchronous process* can handle the computationally intensive work outside the feedback and control loop's sequential nature, prior to receiving the next data point to reduce the anomaly detection computational part during analysis to comparison only, while data streams are blocked.
The expected value ranges can also already be validated and verified against static rule-sets defined by domain experts for desired behaviour. If any part of the expected value range violates these rules, the ranges are adjusted accordingly to adhere to these boundaries. Diagram 7.3 illustrates how the analyse phase compares real values collected with already provided predictions generated by a separate asynchronous process, which continuously calculates expected values through regression tasks, pattern recognition, and data trend analysis. In return, this asynchronous process receives the current data point in every loop to enhance prediction precision for future expected values.
If one or more of the current values violate the threshold range of its expected counterpart, it indicates an anomaly. Additionally, if the predicted system classification does not match

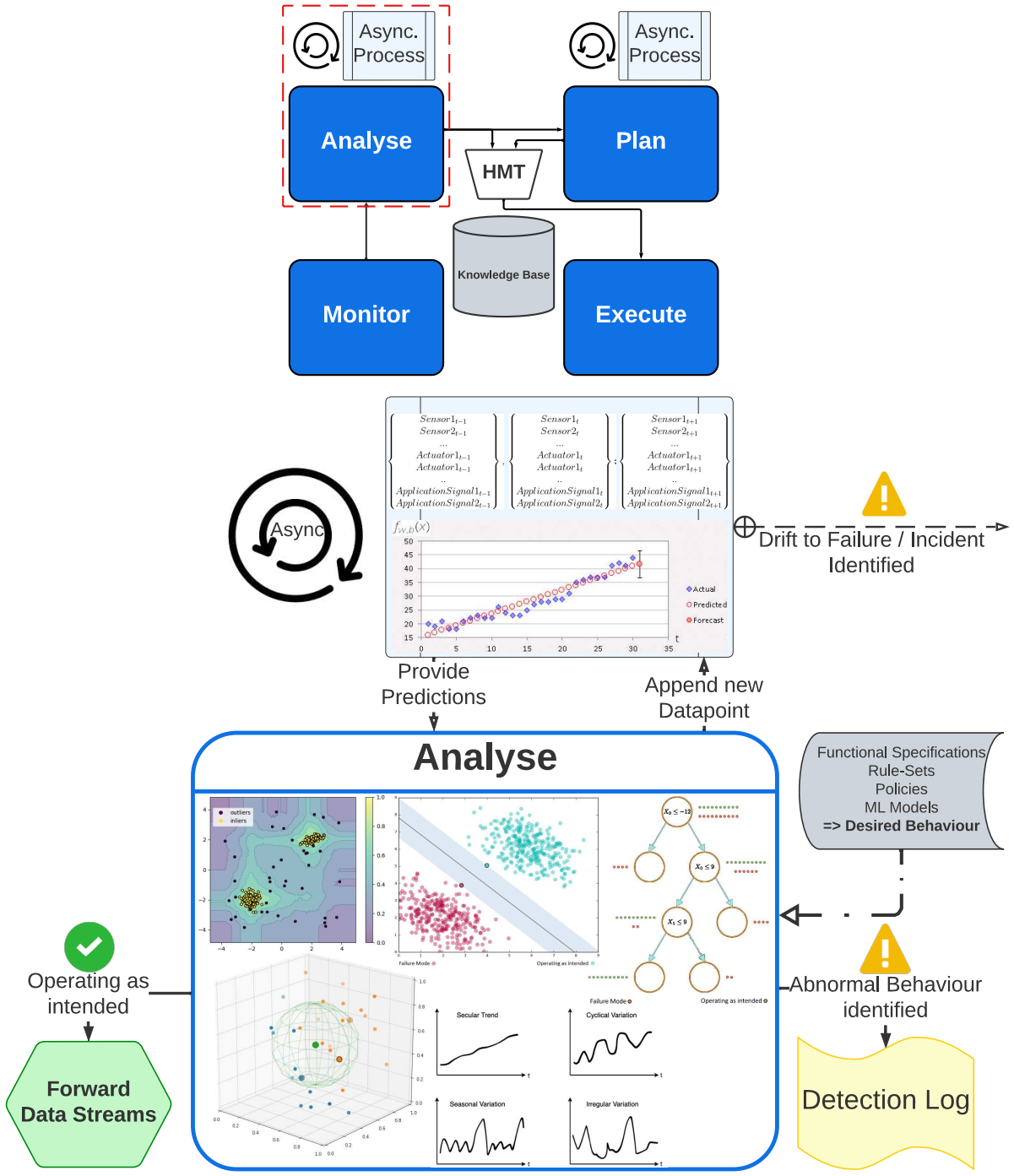


Figure 7.3: Architecture - MAPE-K Analyse Phase
Pictures: [36], [40], [37], [38], [41], [18]

the current collected values, and the system state would be classified as not operating as intended, it would be decisive for the analysis identifying abnormal behaviour. Furthermore, the system-wide classification can also be supported by regression tasks. In the k-NN model or the linear regression model, the system state is being represented as a data point at any given time. When this point approaches the boundary (in linear regression the straight line and in k-NN the sphere boarder) over several time steps, indicating a gradual drift towards failure, the protective shell can anticipate such negative behaviour before it occurs and *proactively plan interventions*. The same applies to decision trees with slowly decreasing path depths, indicating a drift towards failure.

There are numerous analysis mechanisms available for utilisation by specific protective shells, which will be discussed in more detail in section 7.1.3. These mechanisms provide distinct insights into the system's behaviour and collaborate to determine whether the system is operating as intended.
The analysis phase results in a binary decision: either no abnormal operation is detected, and the protective shell assumes the CPS is functioning as designed, or an anomaly is identified, prompting the protective shell to conclude that the CPS is not operating as expected.

Case "operating as intended": The *majority* of results in the analysis phase should indicate that the monitored, embraced CPS is operating as intended with no operation flaws that could have been identified, instilling higher confidence in the system's ability to achieve its defined goals. The comparison of predicted and expected values in this "operating as intended" confirmation from the protective shell control loop is highly efficient and introduces negligible latency in the CPS feedback loop.
Additionally, the system-wide analysis, classifying the current CPS state as either "desired" or "abnormal" behaviour, must be efficient. Engineers have to optimise this classification task's efficiency by selecting a suitable machine learning model tailored to their CPS characteristics.
This outcome represents the base scenario for a properly implemented protective shell and CPS combination, due to the rarity of unexpected, emergent behaviour and its deeply hidden vulnerabilities. It is of upmost importance to forward the interrupted data streams as quickly as possible. Diagram 7.1 illustrates how this "operating as intended" case shortcuts the loop by immediately executing the forward action and returning to the monitor phase, bypassing the planning and execution phases. However, it is possible to strictly adhere to the MAPE-K architecture by first planning the forward action and then executing it in the appropriate phases. As this makes no functional difference, it is presented in the shortcut manner in the architecture overview 7.1 to emphasise the need for efficiency in this step.

Case "abnormal behaviour identified": The second, *rarer and possibly dangerous scenario* involves the analyse phase concluding that the CPS is exhibiting abnormal behaviour, which is safety- and/or security-critical and could pose direct risks if left unattended. This decision should not be taken lightly, as it directly affects the CPS's feedback loop, either by manipulating the data or at least by prolonging the exchange time for such data streams. Therefore, protective shell engineers must thoroughly understand the implications of this analysis decision on their specific CPS and evaluate, during the design phase, the potential impacts of operation delays. Depending on this evaluation, the protective shell may decide during the analysis phase that the detected deviations from desired behaviour are not significant enough to be labelled as abnormal behaviour. In such cases, further investigation is required, involving additional detection mechanisms to fortify the evidence or confirming that the CPS is operating as expected. *This underscores the importance of the collaborative effort among various detection mechanisms to achieve a high certainty score for this critical decision*.
When indeed abnormal behaviour is identified, the protective shell generates the required detection log for subsequent forensic analysis and transitions to the planning phase.

Plan

In the plan phase, adaptations and interventions in the system's behaviour are planned with the overall objective of returning the CPS to a trusted operational state and provisionally mitigate any negative impact while the system operates outside its intended scope. The analyse phase communicates the identified abnormal behaviour to the plan phase, often

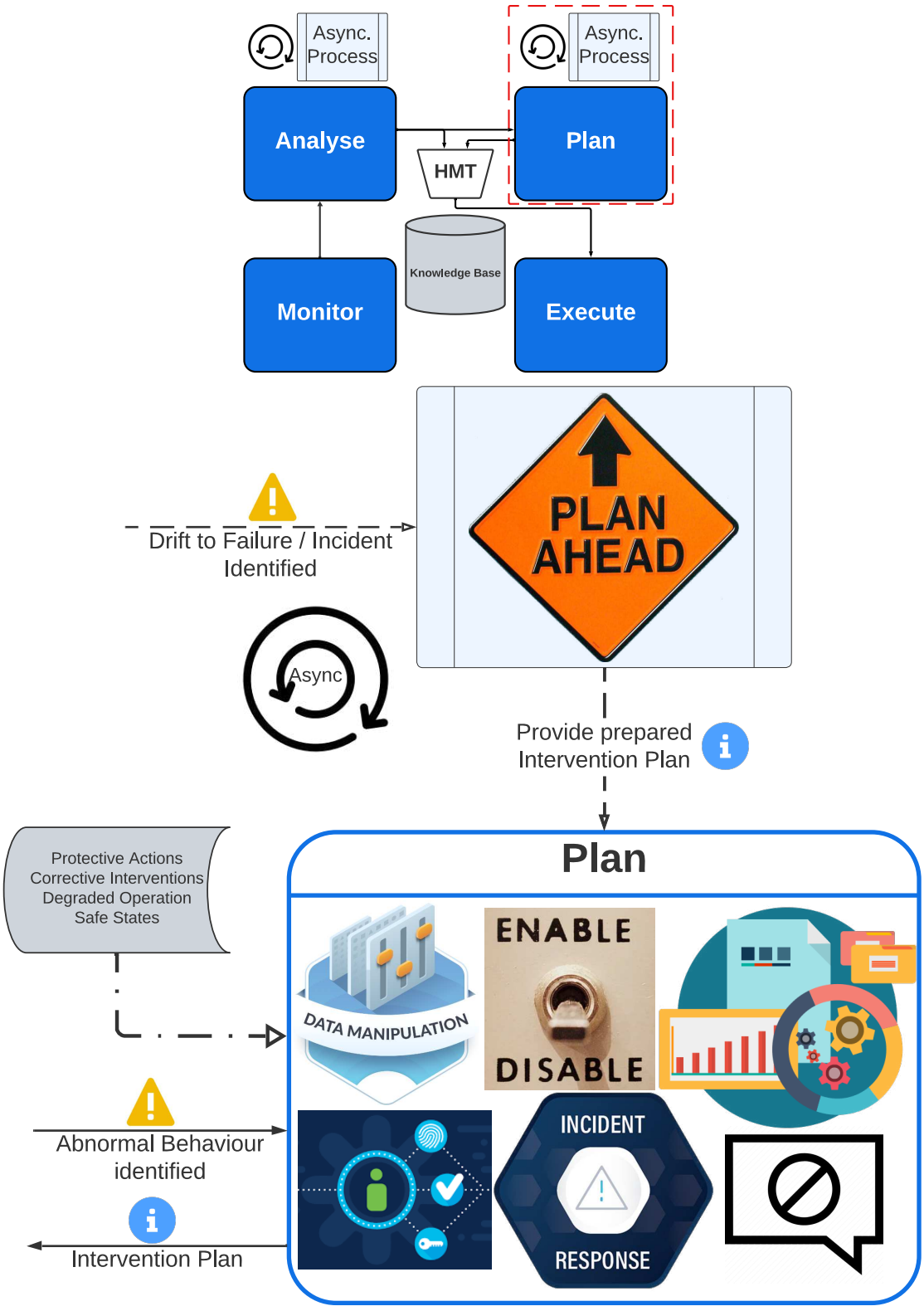


Figure 7.4: Architecture - MAPE-K Plan Phase
Pictures: [43], [44], [45], [46], [47], [48], [49]

already indicating the affected area the anomaly manifested and providing initial insights into the factors that may have led to the situation.
Similar to the analyse phase, the planning phase utilises an additional asynchronous process to reduce the computational workload during control loop iterations. This process prepares

and plans intervention strategies for likely occurring scenarios of abnormal behaviour. It gets notified by the asynchronous analyse process when the system is gradually deviating from desired behaviour, approaching classification boundaries, or when a data stream is consistently moving toward its threshold limits. These scenarios provide the basis for forecasts that the CPS might soon not be evaluated any more as "operating as intended", initiating the preparation of intervention plans outside the planning phase. It plans ahead intervention actions for the most likely occurring abnormal scenarios provided by the analyse regression task evaluation.
Therefore, when the control loop reaches the planning phase, it first searches for already existing intervention plans for the current abnormal scenario. If a matching plan already exists for a highly similar scenario, it can either be immediately published to the executing phase or quickly adjusted to better align with the current system state. Consequently, this *minimises the latency introduced when creating an intervention plan* from scratch and helps efficiently mitigate impending failure modes or limit damage potential during a security incident.
In cases where no prepared plan is available, the intervention actions have to be planned during the sequential control loop. This process mirrors the asynchronous process, but now incorporates specific inputs related to the identified abnormal behaviour. The planning phase can create actions targeting faulty components for deactivation or counteracting the negative effects they introduce into the system. For example, if the analyse phase identifies a malfunctioning AOA sensor, akin to the incidents involving the Boeing 737 Max, the planning phase plans a protective action to deactivate the MCAS system or a corrective action to recalculate the angle using correlated data streams and override erroneous values.
While identifying the abnormal behaviour sources, the Hitting-Set-Tree algorithm is useful and will get presented in section 7.1.3.
In scenarios involving vulnerability exploitations and malicious threats to the CPS, the planning phase establishes active intrusion responses, such as filtering suspicious packets, automatically isolating dubious tasks, or enforcing strict identity and access management by blocking abnormal user actions.
Furthermore, the planning phase can choose to transition the CPS's operation state to a *degraded mode*, restricting complex and potentially higher-risk functionalities in the current scenario. If predefined, the intervention plan may include actions to fall back to a *safe operational state* or revert to a *safe operational envelope*. Moreover, in scenarios where safety- or security-critical tasks are malfunctioning or were suspended, the intervention plan encompasses actions to safely restart those tasks or processes. In severe situations where the planning phase cannot identify suitable counteractive measures and sufficient time remains, the intervention plan may include *handing over to manual control by the CPS operator*.
Additionally, the planning phase can access the underlying knowledge base to retrieve predefined and common reaction patterns for likely scenarios that can be mitigated with simple steps, such as blocking out-of-range actuator values or interrupting startup when code integrity checks fail. The protective shell must have a means to rapidly assess the severity of abnormal behaviour. If the impact on the trustworthy operation of the CPS is minimal, corrective actions don't need to be crafted as precisely, favouring the lower computational effort, thus lower latency, of simpler interventions that would otherwise threaten system stability due to slow control loop iterations. However, in high-severity situations where safety- or security-critical components are affected, protective actions must be rigorously assessed and validated for their effectiveness, reducing the risk of interventions that make the situation worse. This requires additional computational effort and, consequently, higher latency.
Once the intervention plan is prepared, it is typically sent directly to the execute phase to initiate the planned actions. However, in the adapted MAPE-K architecture of the protective shell, the intervention plan is an essential part of the *human machine teaming* (HMT), which

will be discussed in Section 7.1.2. Therefore, it first undergoes collaboration with the CPS operator during this step, facing additional decision processes before being passed to the execute phase.

Execute

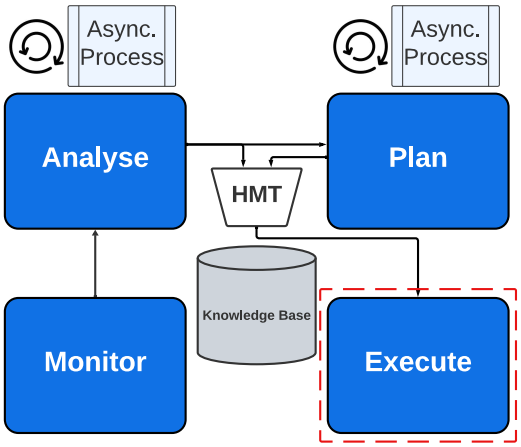


Figure 7.5: Architecture - MAPE-K Execute Phase

The execute phase executes planned actions without introducing additional logic.

7.1.2 Human Machine Teaming

The human machine teaming (HMT) paradigm is specifically engineered to promote collaborative partnerships between human operators and autonomous machines. It goes far beyond the degree of collaboration expected in human-on-the-loop and human-in-the-loop systems, emphasising *interactions, partnerships, and teamwork between humans and machines* [42].

In the context of safeguarding safety- and security-critical CPS, the basic MAPE-K reference architecture provides for fully autonomous monitoring and adaptations of the embraced CPS. Nevertheless, it is imperative to account for the human operator's role, who often directly interacts with the CPS or periodically supervises its operations. The human operator possesses additional information sources, experience, and contextual knowledge, making it crucial to consider his observations and actions when striving to ensure the trustworthy operation of the CPS.

Rapid advancements in artificial intelligence, inherently supported by MAPE-K's knowledge base, have shifted the focus from traditional human-directed robots to fully autonomous ones that do not require explicit human control and take full advantage of machine autonomy to perform tasks independently, efficiently, and quickly [42]. However, there have been also made notable advances in HMT, capitalising on the respective strengths of both humans and machines, while compensating for each of their potential limitations [42]. There exist three key attributes that facilitate effective HMT: *Transparency*, *Augmented Cognition* and *Coordination*.

Transparency is concerned about the observability and predictability of machine tasks, human actions, and their respective intentions.

Augmented cognition encompasses the adaptations of the CPS, safeguarded by the protective shell, while ensuring that the operator remains informed of critical issues and abnormal behaviours through alerts. Furthermore, it employs a shared solution space, allowing both the

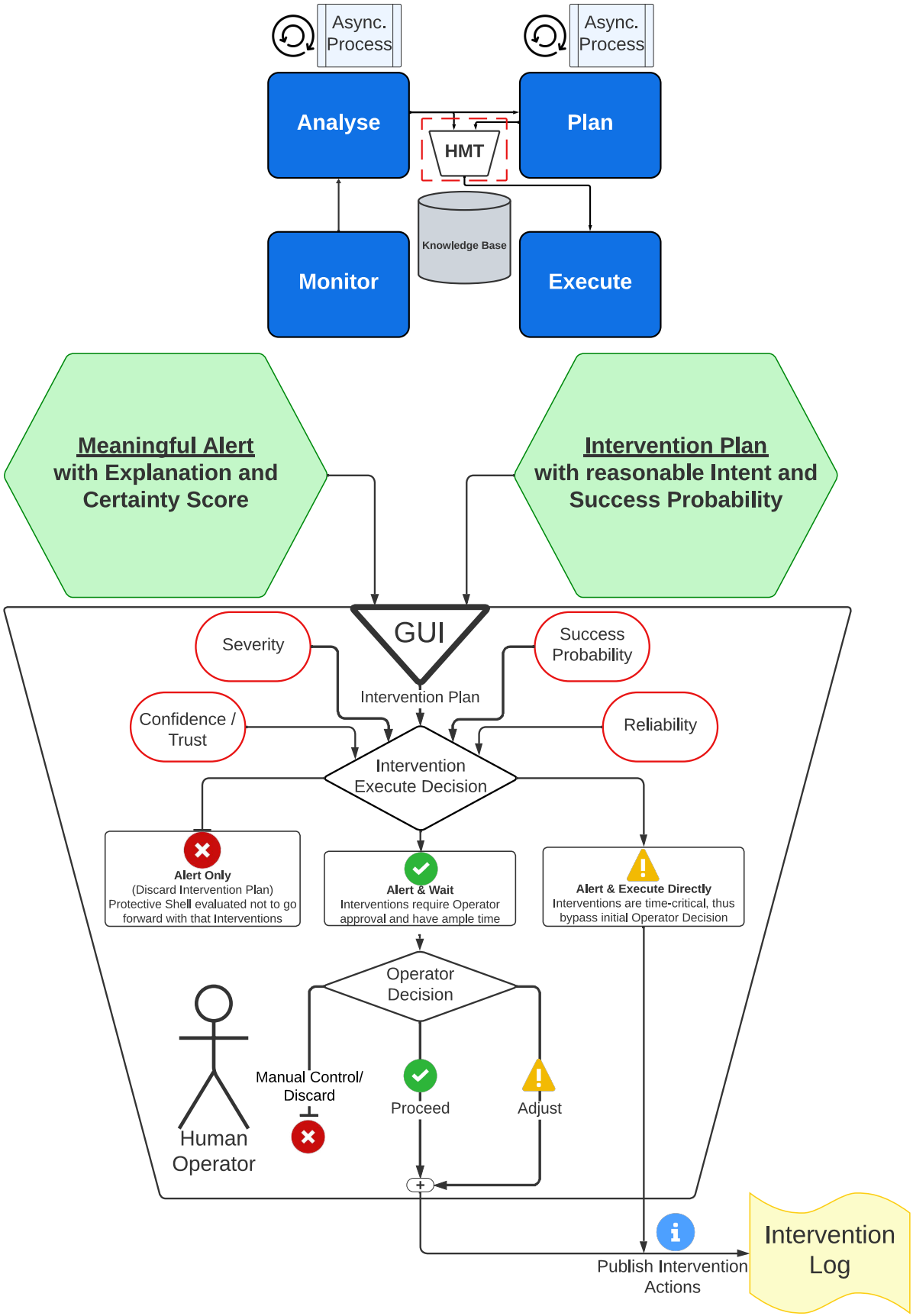


Figure 7.6: Architecture - Human Machine Teaming

machine and the operator to suggest interventions, enabling collaborative problem-solving and finding a common ground.
Lastly, *coordination* provides the operator with the means to override the protective shell's decisions and authoritatively dictate actions to be undertaken. Nevertheless, coordination also injects trustworthiness into the overall system, increasing the operator's confidence when working with the CPS. Trust metrics like the certainty score of an alert's authenticity or the success probability of an intervention plan play a significant role in this context.
As depicted in figure 7.6, the protective shell architecture incorporates HMT after the analysis phase and the planning phase. When abnormal behaviour is detected, the protective shell automatically alerts the human operator, providing information on the cause, region, a certainty score, and a detailed explanation that illustrates the analysis process mechanisms and results, as depicted in the GUI (Graphical User Interface) in figure 7.7. Subsequently, after the intervention action planning phase in the background is complete, the plan is appended to the existing alert on the operator's GUI when the protective shell determines it suitable for publication. Otherwise, if the plan does not meet certain criteria, it will be discarded, and the operator will only be alerted.
It is essential to recognise that the primary aim of any feedback / control system is to reduce human involvement by removing them from the loop, emphasising autonomous decision-making and self-adaptation while maintaining the autonomous characteristics of the CPS itself, under the protective shell's safeguard [42]. Therefore, the two decision steps delineated in figure 7.6 are crucial.

Protective Shell Execute Decision

Initially, the protective shell evaluates its prepared intervention plan by considering attributes such as the severity of abnormal behaviour in terms of its impact on safe and secure operation, the success probability of the intervention actions, the reliability of anomaly detection and planning mechanisms, and the overall confidence and trust within its predictions.
For instance, if abnormal behaviour is detected with limited influence on the current trustworthy operation of the CPS, and the protective shell cannot leverage highly promising mitigation techniques, it may decide to alert the operator and discard its intervention plan. Conversely, in cases where abnormal behaviour significantly impacts the CPS's safe or secure operation and the protective shell can plan effective protective actions with a high probability of success, the decision may be made to bypass the operator decision and *directly* execute the planned interventions. The operator alert and the display of executed or ongoing intervention actions remains untouched and can be viewed on the GUI, with the option for manual intervention or, if feasible, to revert made changes.
However, the majority of protective shell execution decisions should lead to the middle path in figure 7.6, referred to as "Alert & Wait". This decision encompasses scenarios where the protective shell has established a suitable plan that does not require immediate intervention. In such cases, the protective shell can suggest intervention actions to the CPS operator, benefiting from collaboration through manual reassessment and validation. Hence, plans with lower success probabilities, lower prediction reliability, or overall moderate trust levels greatly benefit from manual processes to establish an overall effective intervention that can be executed with higher confidence later on.

Operator Execute Decision

The second decision process centres around the operator's analysis of the situation and the evaluation of proposed interventions. Three possible outcomes emerge: discarding the plan and/or taking manual control, approving the interventions as presented, or adjusting

the interventions before approval. When the operator decides to *discard the plan*, it may be because he did not detect the abnormal behaviour himself and thus distrust the protective shell's findings, he evaluated that the proposed actions do not contribute to restoring the CPS's trustworthy operation, or he assessed that taking manual control can mitigate the dangerous situation more swiftly and reliably.
On the other hand, when the operator chooses to *approve and proceed* with the intervention plan, it indicates that he either identified the same abnormal behaviour or trusts the protective shell's detection based on the provided in-depth explanation. Additionally, it shows that he also believes that the proposed actions are suitable for remedying potential hazardous scenarios.
Lastly, the middle ground allows the operator to *adjust* the presented intervention actions before approving their execution.

GUI

To facilitate the cooperation between the operator and the protective shell, a suitable GUI is indispensable. This GUI should provide an exceptional user experience (UX) by simplifying information access for the operator and making interactions as effortless as possible while offering all essential functionalities required for effective HMT. As illustrated in figure 7.7, a GUI suggestion is presented, populated with alerts and linked intervention actions based on the Boeing 737 Max safety accident example 5.1 to offer a conceptual representation of its operation. It's essential to clarify that this suggestion does not aim to provide the optimal user-friendly solution; rather, it serves as a foundation showing basic functionality that can be provided. Nonetheless, this foundation should be further improved through focus group interviews and usability testing, representing considerations that are outside the scope of this thesis.
Furthermore, it's important to differentiate between the two primary types of operators in CPS HMT scenarios:

Professional Operators These individuals possess in-depth knowledge and training in the specific CPS they operate. For instance, an aircraft pilot falls into this category. They are well-equipped to delve into discovered abnormal behaviour, understand intervention plans, and even provide their own manual protective actions. Therefore, the GUI for professional operators should cater to their need for *detailed information and control*.

Casual Operators Casual operators, like car drivers in semi-autonomous vehicles, often have limited expertise in the system's operation. They require a simplified view with minimal options and clear instructions. Casual operators are less likely to provide their own analysis or actions, let alone feel comfortable to adjust provided intervention plans. Their preferences typically lean towards straightforward choices, such as being alerted and having options to just accept or refuse proposed interventions. Hence, the GUI for casual operators should prioritise *simplicity and ease of use*.

This distinction highlights the significantly different expectations and requirements of these two operator types when it comes to GUI design.
Concluding, in the design of the GUI for CPS, HMT should consider the specific needs and expertise levels of the operators who will use it. This consideration will be included as a design principle in section 7.1.4.
The GUI presented in following figure 7.7 is target towards professional operators, as it is based on the Boeing 737 Max example, given its complexity and the range of available options.

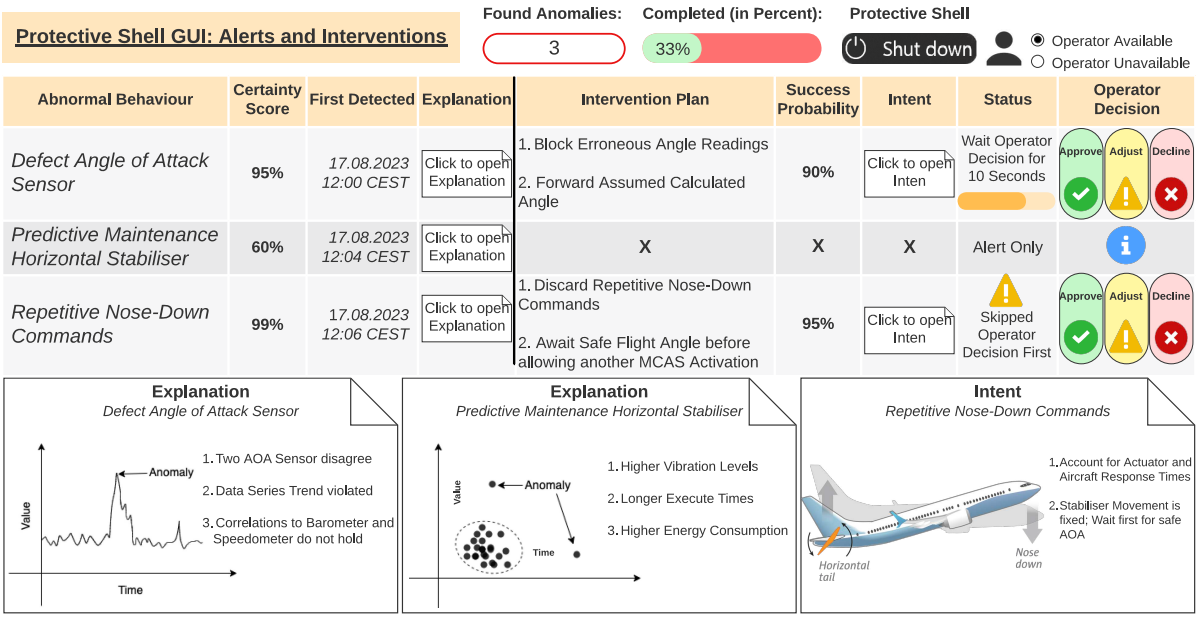


Figure 7.7: Architecture - Protective Shell GUI
Pictures: [18], [50]

On the left side of the table are the alerts categorised according to the detected abnormal behaviour. Each alert includes a certainty score, indicating the reliability of the detection, and is sorted based on the time of detection. This sorting approach results in a static overview, preventing the continuous rearrangement of alerts when sorted by severity during hazardous scenarios, which could lead to operator confusion by loosing overview. In this software-supported analysis, the operator analysis a representation of the CPS's behaviour and/or state through the GUI.
HMT faces the challenge that humans require prolonged time to understand and analyse information for informed decision-making, hinting to static system snapshots that, unfortunately, become outdated rapidly. Therefore, GUI support for HMT should include *dynamic run-time views that continuously update to reflect the current system state and historical information about past behaviour*, rather than presenting just static snapshots of the system [42].
Furthermore, each alert is accompanied by an explanatory report that can be opened to assist the operator's manual evaluation of the situation by providing in-depth, comprehensive insights into the detection process reasoning. Two examples of such explanations are depicted in the lower half of figure 7.7. For instance, the explanation for the defective angle of attack sensor could include a data series diagram displaying the abrupt change in angle readings, which is unrealistic in real-world scenarios. Additionally, it could also picture both angle of attack sensor reading streams side by side to illustrate the deviation and therefore the disagreement between them.

On the right side of the table are the linked intervention plans aimed at restoring the system's current behaviour to its predefined desired state. It initially lists the actions to be executed sequentially, along with a success probability indicating the predicted likelihood of these actions alone remediating the abnormal behaviour or significantly reducing its negative impact on the CPS. Similar to the alerts, a report detailing the intent behind the intervention plan is attached to provide the operator with insights into the protective shell's rationale for selecting these actions. An example of such intent reasoning is presented in the lower-right corner of figure 7.7, justifying protective actions. The protective actions of discarding repetitive nose-down commands and the time-out period before allowing addi-

tional MCAS activations are justified as follows: Repetitive nose-down commands are not accounting for the delay between command execution moving the horizontal stabiliser and the passenger aircraft reflecting those changes. Furthermore, if the aircraft's AOA does not reduce to a safe range after executing the nose-down command, there must be something wrong with the horizontal stabiliser or the sensors. Continuously, executing non-functional commands does endanger the system's safe operation without providing benefits.
These justifications serve to clarify why these actions were chosen and how they contribute to the system's safety.
Moreover, in the two right-most columns, the current status of the intervention plans is displayed, alongside buttons for the operator's decision-making. The status may indicate, for example, that the protective shell is waiting for a defined amount of time for the operator to decide before automatically executing the intervention actions. Alternatively, it may indicate that the operator's decision has been initially skipped but still allows for approval, adjustment, or rejection of the intervention plan at a later time.
Above the buttons, the operator has the option to "log off" from supervising the CPS. While this feature may not be applicable in scenarios like the Boeing 737 Max, where constant human supervision is indispensable, there are CPS that do not require continuous human oversight. This switch assists the protective shell in its decision-making process, determining whether to skip operator decision-making if there is no operator present and the timeframe for executing the protective actions is limited. In such cases, the protective shell might decide to directly bypass the operator decision to ensure timely responses.
Next to the "log off" radio buttons, there is an additional button intended for *emergency situations*, allowing for the immediate shutdown of the protective shell. While a more comprehensive explanation of the rationale behind this feature will be provided in subsequent sections, notably in section 7.1.4 where design and architecture principles are enumerated, as well as in section 7.3.1, which outlines new failure modes introduced by the protective shell, its primary purpose is to swiftly deactivate the protective shell in instances where the protective shell itself encounters run-time errors and negatively impacts the embraced CPS. Consequently, when an operator activates this button, the entire functionality provided by the protective shell is promptly disabled, and all data streams are directly forwarded to their respective receivers, replicating the scenario as if no protective shell were implemented at all.

Nevertheless, a decision is ultimately reached regarding whether or how to execute the intervention actions through the presented HMT process. This decision, along with accompanying explanations, intent, and linked data, is extensively recorded in the according *intervention log*. Subsequent post-analysis of this operational data allows for the determination of why abnormal behaviour was identified initially and how intervention actions were collaboratively planned. Consequently, in the event of a safety accident or security incident, it becomes possible to ascertain the degree of responsibility held by the protective shell, the operator, or whether the fault lies beyond the purview of both.

In cases where operators intervene with the protective shell's actions by reconfiguring objectives or plans, or when they take over manual control, the potential for a *tug-of-war* scenario arises, where humans and machines create competing plans [42]. For example, in the Boeing 737 Max safety accident, the pilots and the MCAS had conflicting plans of letting the aircraft ascend and pulling the nose down respectively, resulting in two crashes. To mitigate such situations, operators can always override initiated actions by the protective shell, as presented previously, and have in the worst case the capability to shutdown the protective shell completely. More detailed considerations regarding this tug-of-war aspect will be discussed in section 7.1.4 regarding architecture and design principles.

Through this rigorous decision process, the three key attributes required by HMT - transparency, augmented cognition, and coordination - are manifested in the protective shell architecture. This establishes the foundation for effective cooperation between the operator and the protective shell, *yielding the associated benefits of effective collaboration*.

7.1.3 Protective Shell Plugin Catalogue

The two subsequent sections provide *plugin lists* for both *detection mechanisms* and *interventions*. Engineers can select plugins during the development process that align best with the specific characteristics of their CPS. This approach enables modularity within the protective shell framework, enhancing code reusability and quality by leveraging open-source, peer-reviewed code bases. Additionally, it reduces the work efforts in implementing protective shells across various projects by utilising pre-existing plugins.
As stated several times previously, it is crucial to configure the protective shell for its embraced CPS to efficiently operate, make accurate predictions, and propose suitable interventions. Failure to adjust the protective shell accordingly can introduce operational risks and hinder its ability to fulfil its objective of enhancing CPS trustworthiness during run-time. Hence, the existence of a plugin catalogue is essential and should be expanded over time.

Protective Shell Detection Mechanisms

The protective shell's detection mechanism list encompasses all determined methods for identifying abnormal behaviour discussed in this thesis. By combining multiple techniques, the protective shell can leverage the true potential of anomaly detection during the analysis phase. It calculates a certainty score by consulting individual mechanisms, overlays their findings, and ultimately integrates the results into a comprehensible explanation.

Functional Specification Divergence Functional specifications are established by domain experts and engineers, outlining the system's desired behaviour. These specifications, although relatively static, set the broad boundaries for the system's functionality. A vital functional specification for any cyber-physical system involves range validations for sensors and actuators. Those are special types of input-output relationships that can get defined through functional specifications and enforce safe actuator operation within predefined limits and specify unattainable sensor readings.
Additionally, for CPS that have various operating modes or states, functional specifications assist the protective shell in acquiring knowledge about possible state transitions, associated actions, and processes.
Furthermore, functional specifications describe anticipated interactions with external systems, delineating protocols and used interfaces.
Functional specifications can encapsulate various other CPS attributes to define intended behaviour. The protective shell utilises these specifications to monitor CPS operation, and determines if it can identify any divergence from the defined desired behaviour. This monitoring helps to ensure that the CPS adheres to its intended purpose, operates reliably, and meets stakeholder functionality expectations.

Rule-sets Violation Rule-sets form the most fundamental operational model of the CPS and are manually crafted by domain experts and engineers. Their stringent nature permits the establishment of directives that must always be followed. Rule-sets find utility in defining event handling procedures, control logic, and ensuring adherence to safety and security constraints.

Additionally, rule-sets play a crucial role in safe and secure system operation, employing safety interlocks, shutdown procedures, access control rules, encryption protocols, and more. They facilitate decision-making by guiding the CPS in selecting appropriate actions based on specific criteria.
Furthermore, rule-sets can define quality of service requirements for the CPS, for instance in the healthcare domain where availability and reliability requirements are crucial. Through criteria based rules, the CPS is guided in selecting the most appropriate course of action.
The protective shell leverages these rule-sets to evaluate whether the current CPS operation violates established rules, indicating abnormal behaviour. Rule-sets ensure the CPS responds adequately to a range of conditions and events, all while achieving its objectives. They provide engineers and domain experts with a structured means to define CPS desired behaviour.

Policies Compliance Issues Policies, often formulated at the management level of organisations or by regulators, define industry-specific guidelines that CPS operating within a given sector must adhere to. For instance, the aviation domain has strict regulations and policies in place that CPS must comply with to reduce the risk of aviation operations to an acceptable level. Regulations such as the Federal Aviation Regulations (FARs) from the United States or the European pendant from the European Union Aviation Safety Agency (EASA) regulations. Moreover, various standards and certifications established that all contribute to safety, security, and additional system quality attributes.
On an organisational level, CPS manufacturers can voluntarily establish policies like ethical use policies restricting morally questionable misuse of their systems, security update policies ensuring that systems aren't running outdated software versions, or privacy policies which extend the mandatory privacy regulations compliance.
The protective shell incorporates policies that are feasible to validate during run-time. If compliance issues arise it directly indicates that the CPS is not operating in accordance with engineers' intentions or regulatory mandates.

Defying Correlations CPS often exhibit correlations among sensor readings, application signals, and other data streams. These correlations may stem from physical laws or software-defined relationships. Engineers and domain experts are responsible for retrieving as many of such faithful correlations for their system as possible, assessing their plausibility, and making them accessible to the protective shell for continuous verification. Additionally, the protective shell can introduce supplementary sensors to establish additional correlations and gain insights into the inner working of the CPS in areas where it suffers from information scarcity.
For instance, in the Boeing 737 Max's MCAS, a correlation exists between the horizontal stabiliser and the angle of attack. When the horizontal stabiliser moves, the angle of attack should subsequently adjust shortly after. Some correlations may be complex, spanning from several data streams and are likely not manually retrievable. In such cases, training machine learning models with hidden inherent knowledge in the training data can aid in uncovering these correlations, as elaborated in more detail later in section 7.1.3.
Nevertheless, sudden deviations from established correlations indicate the presence of abnormal behaviour and guides the protective shell toward its error source. Establishing and validating correlations is essential for the trustworthy operation of CPS, while also testing the engineers' comprehension of their own system's inner workings.

Data-series Analysis Individual data streams often exhibit variations and trends throughout their reporting periods. For fundamental variations or trends within the CPS, it is valuable

to persist them in the desired behaviour for the CPS. Furthermore, similar to correlations, machine learning models can aid in identifying more concealed trends and variations that might get overlooked by domain experts and engineers.
Data-series analysis, synonymous with time-sequential analysis as often referred to in chapter 5, evaluates whether current data series deviate from their anticipated trends or variations. When a data stream significantly diverges from its expected counterpart, it indicates the protective shell abnormal behaviour. For instance, during a specific flight route, the barometer should exhibit consistent trends across different flights on that same route. Detecting unusual height gain or loss during the current flight signals abnormal behaviour.

Code Integrity Checks Code integrity checks serve two critical purposes. Firstly, they ensure that only trusted code, including the embraced CPS application software, component firmware, and protective shell software, is executed. Secondly, integrity checks also guarantee the completeness and compatibility of required data and information within the CPS.
Code signing procedures safeguards the protective shell and embraced CPS from malicious update requests seeking to bypass security or safety measures by manipulating code that later gets executed. Data completeness and compatibility checks prevent the CPS from running with incomplete installation states, missing configuration data, or incompatible resources. These checks are performed during startup and whenever an update request is detected or configuration data is adjusted, imposing minimal latency during normal CPS operation.
Informing the protective shell about trusted entities, from which signed update requests or configuration changes are considered valid, enables it to rapidly identify mistrusted requests by employing asymmetric cryptographic mechanisms.
In summary, code integrity checks empower the protective shell to ensure that the CPS operates with a trusted, unaltered, complete, and compatible code base, thereby increasing trustworthiness.

Advanced Anomaly Detection Advanced anomaly detection relies on *training machine learning models to establish a detailed behaviour baseline*, complementing manually crafted desired behaviour by the engineers and domain experts, as shown in chapter 6. Particularly through detection mechanisms linked to AI, the protective shell facilitates a shift from conservative reactive mitigation approaches to proactive identification of emerging negative behaviour, providing the first step to counteract potential dangerous outcomes.
Engineers must select a suitable ML model that aligns with their CPS characteristics to ensure accurate and efficient anomaly detection. Linear regression models may be suitable for CPS exhibiting linear relationships among its data streams, while non-linear relationships might require decision trees. K-NN models are ideal for scenarios with large multidimensional data records.
While the choice of ML model largely depends on CPS characteristics, there are essential ML model attributes that are strongly suggested, no matter what model gets ultimately chosen, when taking into consideration for implementing in a protective shell. The ML model should be in the best case inherent interpretable or at least has intrinsic explainability. This is crucial for providing timely explanations to operators during run-time and aiding manufacturers in evaluating operational data during forensic analysis.
Additionally, models should support supervised or semi-supervised training to establish a behaviour baseline guided by domain experts.
The following detection mechanisms *predominantly* rely on ML models and provide the protective shell with the capability to compare current system behaviour to an extensive behaviour baseline holding viable intrinsic information, that is not achievable with static and manually crafted detection mechanisms.

Regression Task Comparison Regression tasks project recorded data series into the future, calculating expected values based on past data combined with trained behaviour for each data stream. They are fundamental detection mechanisms that reduce computational work during the analysis phase and enable early identification of impending abnormal behaviour like drift to failure or first sign of security incidents.
After calculating the expected next average value (for each data stream), the protective shell defines upper and lower limits, forming acceptable thresholds for comparison. How narrow these threshold ranges are depends on the predictability of the data stream, the reliability of correlated data, and the overall accuracy of the model. During analysis, the protective shell compares the real recorded values with the precalculated expected ranges, swiftly identifying unexpected behaviour and providing initial clues to its error source.
To profit from the benefits of regression tasks, calculations must outpace the feedback loop of the embraced CPS, meaning expected value ranges should be computed in less than $\frac{1}{SamplingFrequency}$, where sampling frequency defines how often the protective shell receives new data records per second.

Pattern Recognition and Variance Evaluation Pattern recognition is essential for establishing behaviour baselines for processes not directly tied to numeric values. This includes patterns related to component communications via packets and messages, as well as user login and access patterns within Identity and Access Management (IAM). The protective shell can employ this detection mechanism to monitor network traffic and access, inspecting packet payloads for malicious or known malware signatures. Furthermore, the protective shell can leverage IAM capabilities to evaluate user behaviour and detect significant variances in their current activities or access patterns, such as suspicious authentication requests, abnormal actions, or patterns indicative of privilege escalation attempts to gain elevated access rights.
Patterns can also be learned during ML model training, for example, when different components collaboratively work toward a common goal, they communicate in predictable patterns. Deviations from such patterns or the discovery of entirely new patterns aid the protective shell in identifying abnormal behaviour.

Software Probes Examination Software probes are retrieved during CPS monitoring, providing information from application software and firmware reporting tools. By establishing a behaviour baseline for safety- and security-critical tasks within the CPS, ML models learn typical resource usage, normal response times, and foremost what crucial tasks are running.
Process monitoring enables the protective shell to swiftly detect non-responsive, suspended tasks critical for providing essential functionality. Moreover, it identifies when the CPS approaches its resource limits, potentially resulting in performance degradation upon reaching bandwidth, storage, or computational power thresholds.
Lastly, the protective shell can examine software probes to detect (attempted) intrusions during impending security incidents, limiting attackers' access and preventing further spread through CPS components.

Classification Classification is a key detection mechanism that offers a holistic view of the entire system, instead of looking into specific little details like most other detection mechanisms presented, by considering all accessible data streams and evaluating them based on a pre-trained ML model. This model, trained with (preferably labelled) operational data and log files indicating if this corresponded to a normal operation, or the system experienced a safety accident, security incident or other undesired behaviour.

By overlaying this model, splitting those two statuses apart, as a classification template onto the current system state, the protective shell efficiently determines if the CPS is currently operating as expected or deviating from desired behaviour.
This technique can be further enhanced by incorporating previous classifications into a regression task, projecting the most likely next classification. Consequently, early detection of the system drifting toward undesired behaviour becomes possible. The efficiency of this classification task depends on the chosen ML model and its classification mechanism. Furthermore, it is important for engineers to not only chose a suitable ML model according to the CPS characteristics, but also select valuable training data and apply training mechanisms procedures to reduce overfitting tendencies.

Conclusion In conclusion, engineers and domain experts must choose suitable detection mechanisms to equip the protective shell with the means to identify abnormal behaviour. *While employing multiple detection mechanisms enhances the protective shell's ability to make reliable predictions and increase overall certainty scores, it also necessitates greater computational resources for evaluation, potentially increasing control loop latency*.
Additionally, the use of ineffective detection mechanisms, such as flawed desired behaviour specifications, policies, or rule-sets, or selecting inappropriate ML models for CPS characteristics, can result in inefficient predictions with low accuracy and a higher likelihood of subjecting the system to dangerous and unnecessary interventions. Therefore, specifying detection mechanisms with care, absolving peer-reviews, and rigorous testing is of upmost importance to mitigate the risk of unintentionally implementing logic flaws or incompatible measures.
The subsequent table 7.1 provides a comprehensive summary of the presented detection mechanisms. The first column regarding the view indicates the capability of these mechanisms to identify abnormal system behaviour on a *system-wide* scale, unbound of any association with a singular error source, or, conversely, to identify *detailed* abnormalities linked to specific or limited error sources. The adaptability inherent to each detection mechanism corresponds to the ease with which adjustments can be made to their underlying desired behaviours. For instance, the process of retraining or enhancing a machine learning model is notably simpler by just providing new operation data than the task of manually formulating logical expressions within functional specifications. For computational complexity and implementation effort, relative metrics are given that can vary based on the specific implementation and system requirements.

Name	Detailed / System-wide View	Adaptability	Computational Complexity	Implementation Effort
Functional Specifications Divergence	both	rigid	Low	High
Rule-sets Violation	both	rigid	Low to Moderate	High
Policies Compliance Issues	system-wide	rigid	Low	Moderate
Defying Correlations	detailed	semi-variable	Moderate	Moderate
Data-Series Analysis	detailed	semi-variable	Moderate to High	Moderate
Code Integrity Checks	system-wide	variable	Low	Low
Regression Task Comparison	both	variable	Low	Low to Moderate
Pattern Recognition	detailed	semi-variable	Moderate	Moderate
Software Probes Examination	system-wide	variable	Moderate	Low to Moderate
Classification	system-wide	variable	Moderate to High	Low

Table 7.1: Generic Protective Shell - Overview Plugin Catalogue Detection Mechanisms

Protective Shell Interventions

The protective shell intervention list encompasses all protective and corrective actions discussed in this thesis. Through the combination of several of these interventions, the protective shell can prepare an effective counteracting plan against abnormal behaviour by leveraging its influences on the embraced CPS. By employing multiple actions working cooperatively to guide the CPS back to trustworthy operation, the protective shell can lay out its intervention plan's intent, elucidating how to effectively address the issues found and predicting the probability of success.

Hitting Set Tree Algorithm Nevertheless, after identifying that the CPS is exhibiting abnormal behaviour, it might not be inherently clear where the issue(s) originate. When the detection mechanisms identify single-point anomalies, the error source is obvious. Furthermore, the analysis phase can consult several detection mechanisms and evaluate them by overlaying their results, which will often be enough to narrow down the error area sufficiently. However, there will be cases where these methods are ineffective, and the protective shell is unable to identify a single component or the main contributing parts.
The *hitting set tree algorithm* is a well-established method for evaluating abnormal behaviour identified from formal specifications, rule-sets, and other logical expressions in formal language that represent the syntax of defined desired behaviour cast in several axioms. The algorithm was first introduced by Reiter in 1987 and further refined by Horridge, Parsia, and Sattler in 2009 [51]. Its objective is to efficiently compute all justifications that directly induce the inconsistency, thus revealing the logical flaw that produces mathematical expressions that are unfulfillable once the variables are populated with the current data sets. *Justifications*, in this context, are nothing more than individual explanations for the error the CPS is currently exhibiting.
After finding all justifications, the algorithm also provides an indication which of those explanations, pinpointing a potential error source, is most likely the real cause.
Firstly, the minimal axiom subsets are calculated, which exhibit such inconsistencies on their own from the whole set of logic expressions accessible to the protective shell. This can be done iteratively by taking the entire set of axioms and splitting them in half, while checking if the inconsistency still occurs in one of the split subsets. When that is the case, it gets further reduced until the subset is free of inconsistencies, indicating that one step earlier, the minimal subset was found.
Once the minimal sets of axioms are found, the hitting set tree algorithm constructs *nodes* from these minimal sets of axioms and *edges* from individual data sources. As depicted in figure 7.8, starting from the root, single data sources like sensors, actuators, or signals are excluded and labelled along the edges to the next node. If excluding this specific data source from the formal reasoning resolves the inconsistency, the algorithm stops by showing a checkmark. However, if after excluding this specific source another justification for the inconsistency can be found, it is appended as the next node.
In the next node, the process can be sped up, as indicated by the cross for sensor 1 depicted in figure 7.8, because the data sources that have already resolved the inconsistency in higher levels of the tree don't need to be checked again in other branches. The algorithm completes when no justification can be appended as the next node while still holding this inconsistency.
By examining the created tree, *each path leading to a checkmark indicates a possible explanation*. However, they have varying probabilities of being the real cause of this abnormal behaviour. For instance, looking at figure 7.8, which shows that just by removing the sensor 1 data source from the formal evaluation, the desired behaviour defined by the logical expression holds again. Consequently, by removing Sensor 2, Actuator 2, and Application

Signal 1 from the logical reasoning, the desired behaviour is fulfilled as well. However, both explanations vary significantly in their probability of occurrence. In the first explanation, only one sensor has to malfunction, whereas in the second explanation, it is required that three data sources deliver false data simultaneously. Conversely, the deeper the path until the checkmark, the more unrealistic the explanation becomes.

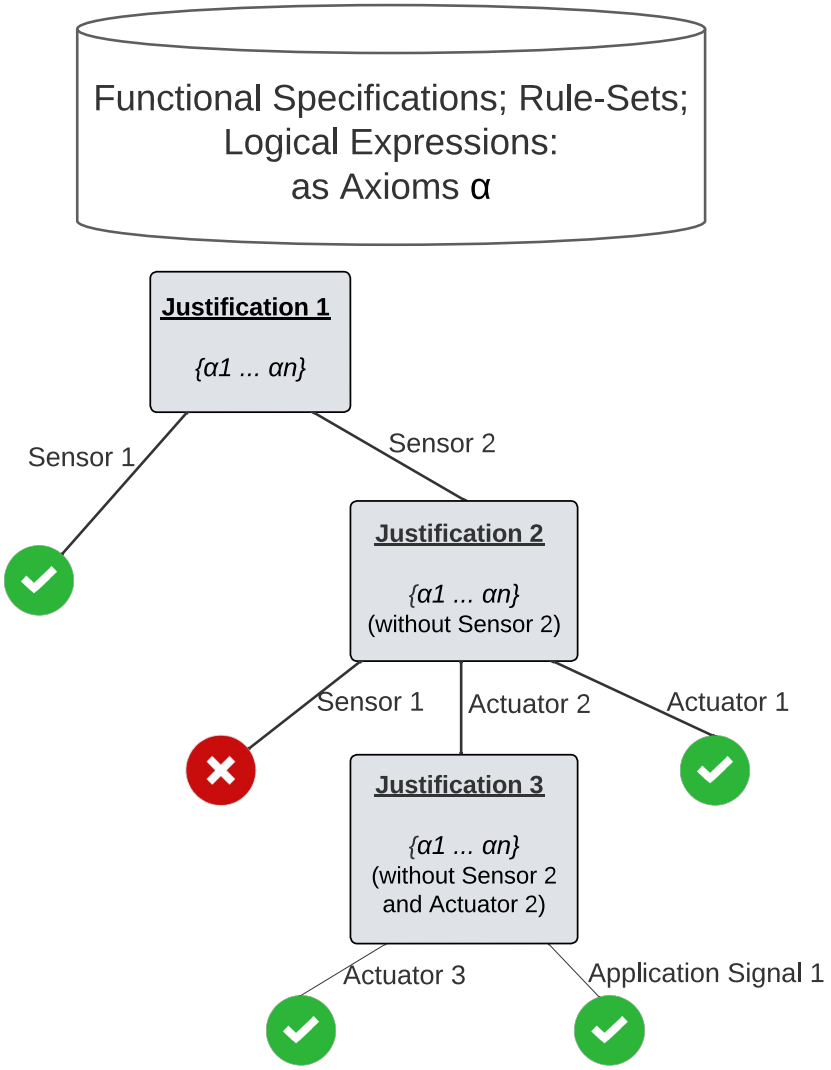


Figure 7.8: Architecture - Hitting Set Tree

Interventions When either during the analysis phase or at the planning phase the source of abnormal behaviour could have been identified, the protective shell is tasked with selecting suitable intervention actions to address the issues. As shown in figure 7.6, the protective shell has to decide between only alerting the operator, alerting and suggesting to the operator an intervention plan needing approval, or alerting and directly executing planned corrective/protective actions.
Obviously, when the error source can't be identified that causes the abnormal behaviour, the protective shell can't initiate purposeful interventions aimed directly at the problematic area and might only create universally applicable actions like going into degraded operation. Therefore, in such scenarios, the most likely decision would be to only alert the operator.
However, in scenarios where the supposed error-causing source could be identified, the protective shell needs to evaluate more deeply which decision path to take. For this decision-

making process, it is important to note that the overall risk of a safety accident or security incident occurring is *not* decisive for whether the protective shell should take action. *Instead, it is the difference between the calculated risk of the impending accident/incident without interventions executed by the protective shell and the residual risk of such an accident/incident after protective measures are initiated*. For instance, just because a safety accident is much more likely after an aircraft's turbine defect does not mean that the protective shell stops providing interventions because of the overall elevated operation risk. The protective shell continues to provide its functionality whenever it evaluates that it provides added value for the trustworthiness of the embraced CPS.
Therefore, during the decision-making process regarding how to execute the planned interventions, it considers a wide variety of linked attributes like the severity of found abnormal behaviour concerning the negative impact it could have on the CPS, the reliability of the information received from data sources, the success probability of the plan, and others. One of the most important factors in the decision if the operator's approval should be bypassed or not is the *estimated available timeframe to react* before the planned actions become insufficient or the impending accident/incident can't be remediated any more. Therefore, if such a timeframe is too short for the operator to analyse the situation and form their own opinion, the protective shell tends to execute the actions directly. However, if such a timeframe is sufficient to consult the operator, a second manual evaluation often proves valuable.

Alerting and Logging Alerting and logging are the only two passive mechanisms employed by the protective shell, besides simply forwarding all data streams when no abnormal behaviour is identified. This means that while executing these mechanisms, the protective shell does not actively change the system's behaviour, manipulate components, or engage in other intrusive interventions directly influencing the CPS.
Meaningful alerts contain the abnormal behaviour found, an explanation often pointing to the error's cause, and a certainty score indicating how confident the protective shell is with this prediction. Extensive logs are always written when the protective shell identifies unusual behaviour or executes interventions. However, the logging can be expanded to also contain additional scenarios if the storage requirements are met according to the detail level and the information frequency.

Override Data Through data manipulation, the protective shell has an effective and purposeful capability to address erroneous sensor readings, invalid application signals, or similar incorrect data streams traversing through it. Overriding with correct values is especially effective, since functionalities within the CPS that rely on these data streams are not affected and can fully serve their purpose. For instance, when a sensor experiences a failure and delivers erroneous readings to the protective shell, the protective shell might detect such incorrect values, calculates replacement values through correlations, and finally overrides them before letting the data pass.

Discard / Block Data Since the protective shell acts as a proxy for communicating with the CPS, where all data streams have to pass through it, it has the capability to discard or block traffic that is identified as unusual and could have a negative impact on the system's operations. However, this is not always the preferred counteractive measure, since overriding the data with trusted values would be more efficient and less intrusive during run-time. Nonetheless, there are scenarios where either the recalculation of data streams is infeasible or unwanted. For instance, when the Therac-25 sends an erroneous application signal for starting a treatment with an out-of-range radiation dosage, the protective shell must not recalculate the appropriate dosage since the chance of forwarding an incorrect value that

does not match the doctor's treatment plan should not be taken. Therefore, in this case, discarding the application signal is the appropriate protective action since there are no cascading negative effects triggered by these interventions and the doctor can re-initiate the treatment after reassessing the parameters.

IAM Interventions Identity and access management interventions are applied while monitoring network access to the CPS and outgoing network traffic. It enables the protective shell to enforce strict access control and aid the identification as well as authorisation process. The most typical example for effective IAM is blocking attackers from accessing the CPS with stolen credentials because of unusual login times or unknown locations. However, the protective shell also monitors user activity within the CPS, which actions get initiated, what configurations are adjusted, or which processes are stopped. When unusual or malicious activity is detected, the protective shell can block the user, restrict them from initiating further actions by temporarily removing permissions, or request additional authorisation factors to verify the current session, as well as ask a third user to vouch for him.
This is best exemplified by looking at the Ukrainian power grid attack, where a few users initiated a high number of circuit breakers to open in a short amount of time, which would likely have been identified by the protective shell as unusual activity resulting in access control counteractive measures.

Packet Filtering / Rate Limiting Packet filtering and rate limiting are intervention techniques employed by the protective shell to typically safeguard the CPS from malicious user actions. It is similar to data blocking and discarding for operational safety, but with a focus on preventing security incidents.
The protective shell detects suspicious packets and messages by pattern recognition, utilising the established behaviour baseline. Packets that do not follow known patterns can be quickly identified and filtered. Furthermore, by scanning the payloads of messages for malware signatures or significant deviations from typical payloads for that message type, the protective shell can actively block and discard such positively reacting messages before they can deliver their possibly malicious payload.
Furthermore, by rate limiting message processing, especially from outside sources or mistrusted addresses, the protective shell can slow down cyberattacks, for example, by making brute-force attacks unfeasible.

Control Processes and Tasks With processes and tasks control, the protective shell employs the capability to rectify unwanted software behaviour and coding flaws. Through software/firmware probes and application signal monitoring, a wide variety of identification techniques can be utilised to detect suspended safety- or security-critical tasks, run-time errors in processes caused by unexpected events and conditions, or suspicious processes executing unwanted functionality, such as during security incidents.
For suspended critical tasks, the protective shell can attempt to initiate an isolated restart affecting only those parts, to prevent a CPS outage while restarting the whole system. Furthermore, when detecting unusual activity by processes hinting at malicious activity, the whole process can be terminated or automatically placed in quarantine. Nowadays, most cyberattacks work with process injection or process hollowing, where the attackers hide their malicious code within pre-existing processes to evade detection and persist on the compromised system. Therefore, just scanning for known processes is ineffective, thus their activity needs to be continuously monitored.

Safe State Fallback All previously mentioned intervention actions are specific and purposely aimed at the error-causing source during run-time. However, there are also universally applicable protective actions that can be executed without knowing the exact cause. Three of them are subsequently stated.
The fallback to a predefined safe operation state is a viable option when the protective shell detects impending safety accidents without finding effective specific counteractive measures. By leading the system back to a safe operational state, the trustworthiness is improved, and the system becomes overall more resilient against single failures.
Such a safe state does not have to be a specific point on the operation scale, but can encompass a whole range of operational conditions that are considered safe. An example of this would be the flight envelope for aircraft with different zones according to their overall stability, as depicted in figure 7.9. Therefore, the protective shell can initiate a fallback from a struggling operational state to a more stable, thus more reliable and trustworthy envelope zone.
Since this safe state definition is very rigid and the universally applicable nature of this intervention, it might not be the most efficient intervention in every scenario and initiates highly intrusive adaptations to the run-time system.

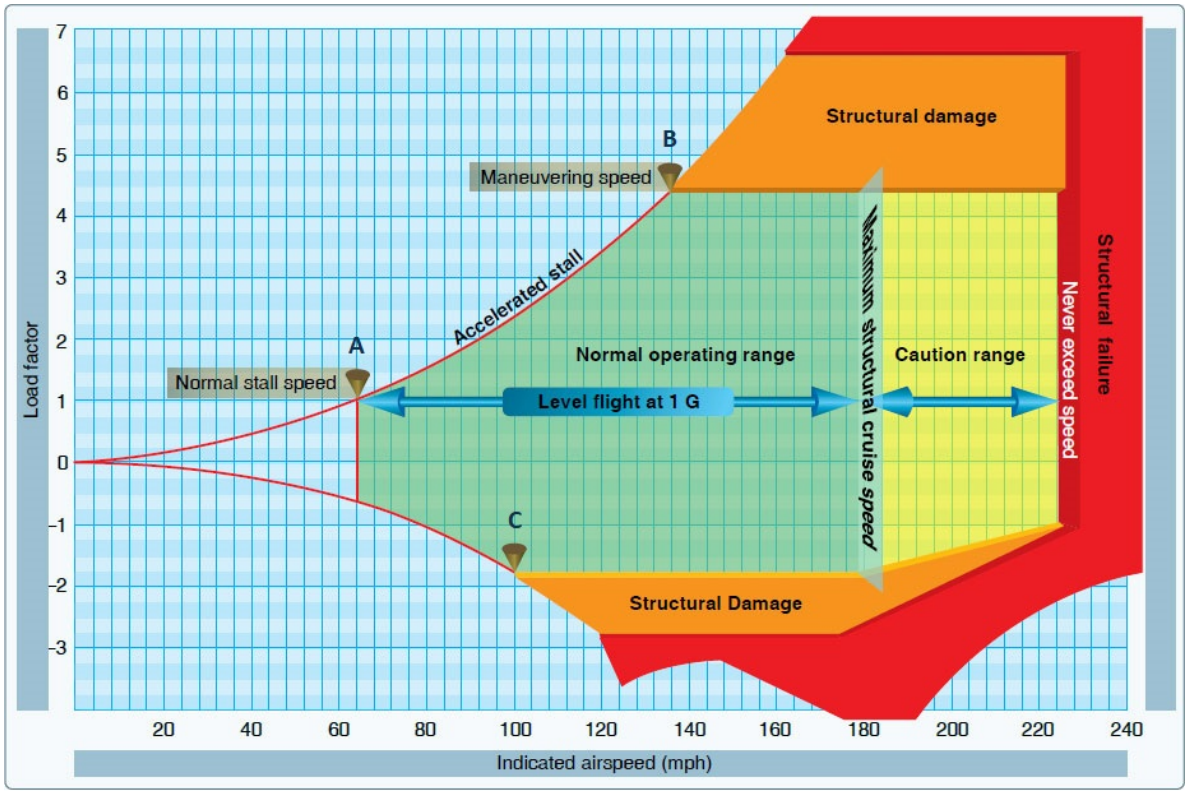


Figure 7.9: Interventions - Flight Envelope [52]

Hand over to Manual Control Handing over to manual control is also a universally applicable intervention for CPS with operators constantly present, who have the means to take over control. When the protective shell lacks large parts of information to provide and ensure trustworthy operation of the CPS, the operator might be asked to temporarily partly or fully govern the system.
This intervention relies on the assumption that operators have access to additional information sources that the protective shell doesn't receive, and that experienced operators can handle hazardous situations better. For example, in vehicle autopilots, it is implemented

that when the weather changes from clear to foggy or rainy, which hampers the sensors from reliable data collection, the autopilot is switched off, and the driver is asked to take over until the environmental conditions improve.

Degraded Operation Degraded operation refers to the scenario where the CPS is not functioning as designed or intended due to various factors such as component failures, unforeseen events or security threats. However, basic functionality is still provided while suffering from this impairing conditions. The protective shell can initiate, as a protective action, going into degraded operations, thereby preventing the execution of complex actions or processes that expose higher risks during hazardous situations.
The main objective is to minimise the system's run-time complexity to a minimum, thereby temporarily reducing failure modes and attack vectors in the CPS. This is one of the most impactful intervention actions presented in this thesis because it affects a wide range of CPS components and applications while also reducing its functionality to a minimum. Initiating degraded operation must be carefully considered and should, in most cases, be approved by an operator.

Conclusion Engineers and domain experts have to define suitable interventions accessible to the protective shell. They should provide a wide variety of tools to lead the CPS, exhibiting abnormal behaviour, back to a desired operational state. Protective and corrective actions presented in this section interfere and manipulate the embraced CPS on different levels, from solely passive actions where the CPS remains untouched to very intrusive actions that seriously could affect the system's stability. That is why it is important for the protective shell to take predictions into its decision-making and planning process, considering whether the interventions ultimately reduce the risk currently affecting the system or make it even worse. Therefore, it would not be optimal to try to interfere with a CPS exhibiting minor inconsistencies that don't pose a great danger to its safe and secure operation with corrective actions that could disrupt normal operation, potentially leading to instability, unreliability, and overall untrustworthiness.
Furthermore, specific, purposeful interventions were discussed that are directly aimed at the identified error source, trying to remediate it, as well as universally applicable interventions trying to bring the CPS into a safe state or degraded operation, where it would be more resilient against threats and failures, whenever an error source with high certainty could not be identified.
Ultimately, it can be said that *providing the protective shell with as many effective intervention techniques as possible for a wide variety of scenarios greatly increases the options it has to address arising issues by crafting effective and highly customised plans* to counteract them. When engineers and domain experts only provide the most fundamental interventions, the protective shell becomes very limited during the planning phase in suggesting mitigation mechanisms to the operator or directly executing those that now might not be as tailored as they could have been.
The subsequent table 7.2 provides a comprehensive summary of the presented interventions. The first column concerning the protective or corrective nature of each intervention indicates whether the action tries solely to *safeguard the system*, or to *remedy the identified issue(s)* by providing a more trusted alternative. The specific or universal type characterises each intervention as either aimed directly at the error cause or as affecting the entire CPS with predefined steps that can be universally applied without context. For example, overriding data streams is highly context-sensitive, requiring knowledge of the exact data source and values for stream overwriting, whereas going into degraded operation does not depend on specific system states or operational conditions. It follows the same predefined steps to transition the CPS into a more resilient mode. The metrics for computational complexity

and impact on the system are provided as relative measures, which can vary based on the specific implementation and system requirements.

Name	Protective / Corrective	Specific / Universal	Computational Complexity	Impact on System
Alerting and Logging	Protective	Both	Low	None
Override Data	Corrective	Specific	High	Low
Discard / Block Data	Protective	Specific	Moderate	Moderate to High
IAM Interventions	Protective	Specific	Low to Moderate	Low to Moderate
Packet Filtering / Rate Limiting	Protective	Both	Moderate	Moderate
Control Processes and Tasks	Corrective	Specific	Moderate to High	Moderate to High
Safe State Fallback	Corrective	Universal	Low	Moderate to High
Hand over to Manual Control	Protective	Universal	None	High
Degraded Operation	Protective	Universal	None	High

Table 7.2: Generic Protective Shell - Overview Plugin Catalogue Interventions

7.1.4 Architecture and Design Principles

To establish a constructive and purposefully crafted generic protective shell framework, it is essential to reinforce it with *sound architectural and design principles*. These principles are crucial for creating a robust, secure, and safe foundation within the overarching protective shell architecture. They serve as fundamental blueprints upon which individual parts of the system are constructed, defining the structure, organisation, and interactions with other components. This enables engineers to conceptualise, design, and implement industry best practices. Furthermore, it allows engineers across different domains to apply this generic architecture framework containing architecture and design principles that directly address safety and security challenges in CPS, making them more trustworthy to operate by offering a standardised blueprint customisable to specific applications.
In summary, there are three main advantages to defining architecture and design principles:

- **Efficiency:** Engineers can expedite the development of effective protective shells by reusing established architectural and design patterns. This saves time and resources while benefiting from their reliability, which has been rigorously evaluated across numerous projects.
- **Consistency:** A generic framework supported by engineering principles ensures a consistent and uniform approach to trustworthiness. This reduces the risk of overlooking critical vulnerabilities or adopting suboptimal safety or security measures.
- **Adaptability:** The generic approach allows engineers to configure and design their protective shell specifically suited for the embraced CPS, improving performance, reliability, and effectiveness in its specific scenario. Furthermore, as the failure and threat landscape evolves across industries, the framework can be updated and further refined, enabling protective shells to provide resilience against emerging negative behaviour in the future.

Subsequently, the architecture and design principles retrieved from the state-of-the-art, stemming from the learnings of developing specific protective shells, found as the requirements towards artificial intelligence, and surfaced during architecture conception will be presented.

Consistently Alert and Log

As previously explained several times before, it is mandatory to always publish meaningful alerts and record detailed logs, which now gets cast into a design principle. Whenever the protective shell detects abnormal behaviour or directly intervenes with the CPS's planned actions, the operator must be informed, and operations data must be written.
During protective shell development, the alerts and the GUI, where they will be displayed, should be adjusted by the engineers according to the user's requirements. Therefore, they should engage with the operators to define what should be contained within the alerts, specifically, which information is important to the operators, and how the user wants to interact with the GUI and what it should look like.
Furthermore, the engineers representing the manufacturer have to define how extensive and detailed the protective shell operation logs should be. While more detailed logs provide more insights into the inner workings, they require more storage and could make outlier detection more challenging since it is hidden within a larger volume of logged data. Additionally, they should define a process that extracts those log files periodically for continuous evaluation purposes and after safety accidents or security incidents for forensic analysis.

Required Range Validations

Therefore, a clear definition of measurement ranges for sensors, operation ranges for actuators, and, where feasible, value ranges for application signals is mandatory. Since any violation of these ranges directly indicates to the protective shell an abnormal behavior with high certainty, these limits must be carefully defined, reevaluated, and tested to limit exposure to specification flaws.
Therefore, a clear definition of measurement ranges for sensors, operation ranges for actuators and where feasible value ranges for application signals is mandatory. Since any violation of such ranges directly indicates the protective shell an abnormal behaviour with high certainty, such limits have to be defined carefully, reevaluated and tested to limit the exposure to specification flaws.

Specify Data Correlations

Engineers and domain experts have substantial knowledge about their field of work and have built up experience regarding which correlations in the embraced CPS exists. Therefore, the engineering principle must be followed of specifying fundamental truths of the system directly in functional specifications for the protective shell to continuously evaluate. Such data correlations might be induced directly by physical laws or coded into the application software. The more of such inherent knowledge of the inner CPS workings are provided to the protective shell, the more accurately it can identify and remediate error causes.
Furthermore, the set of correlations can be extended beyond the manually defined correlations retrieved scientifically, through experience, or coded intrinsically in the software. This can be achieved by considering the correlations that the trained machine learning model has identified. Domain experts and engineers must reassess the model, for example, by examining attention scores, biases, and weight matrices, which directly indicate correlations between different features. If such found correlations from the operations data hold and can be proven in practical terms, they can be directly appended to the correlation set, improving their functional specifications and providing domain experts with new knowledge about their systems.

Describe Data Trends and Variations

Similar to the previous principle, the mandatory requirement of describing known data trends and variations is essential in providing the protective shell with the means to perform time-sequential analysis. Therefore, it is listed as one of the design and architecture principles that engineers should follow during protective shell development and implementation.
Data sources typically exhibit repetitive trends that can be highly valuable for predicting future values. Definitions of such trends tend to have a macro view over a longer time period, rather than trying to provide explanations for short-term fluctuations. With such main data trends and variations, domain experts and engineers are already familiar and can manually craft the corresponding functional specifications. Furthermore, the trained machine learning model might identify deeply hidden yet unknown trends that the domain expert should analyse for formal correctness and either append to their functional trend specifications or discard.
The protective shell's prediction capabilities experience enhanced accuracy with extensive knowledge about inherent data trends and variations exhibited by the embraced CPS, making this design principle essential in almost all scenarios.

Mandatory System Integrity Checks

The system integrity checks principle combines a set of industry best practices that are essential in the CPS domain, greatly enhancing trustworthiness and reliability without any considerable negative drawbacks. The first industry best practice is to apply code signing techniques to the protective shell's application code, as well as software and firmware used in the embraced CPS. This is especially valuable in security-critical domains where the CPS has interfaces actively communicating with external devices or external entities have direct physical access to the CPS. Through code signing techniques employing asymmetric cryptography, the protective shell can reliably detect manipulations to the code base or unofficial update requests not originating from the manufacturer (or any other trusted party), thereby preventing malicious actions aiming to execute non-functional or specifically crafted dangerous code.
Furthermore, the system integrity checks encompass processes to guarantee complete, unaltered, and compatible configuration data and information needed during CPS operation. This is especially essential in safety-critical domains where missing or corrupted data can lead to unexpected behaviour, faults, and ultimately safety accidents. Therefore, these integrity processes confirm the complete and viable installation state of the CPS and its components.
Lastly, the system integrity checks employ procedures to verify that all expected sensors are properly connected and delivering readings, and all actuators are present and responsive to commands, thereby ensuring CPS operability.
All of these system integrity checks are executed once the CPS is started and every time there are changes affecting either the code base, the configuration data, or the CPS components. It is an important design principle for the protective shell, as its objective contributes to enhancing the trustworthiness of the system employed. Without system integrity checks, the CPS might run with a corrupted, manipulated, or incomplete code base and not compatible configuration data, making run-time errors much more likely, or it can't access sensors and actuators from the start, which might limit functionality dramatically.

Predefined Safe States and Degraded Operation Mode

The design principle of specifying a set of safe operation states and defining how a degraded operation mode should look like is essential to equip the protective shell with two universally applicable interventions beforehand: leading the CPS back to a safe operation state and falling back into a degraded operation mode to prepare for impending hazardous situations. During the development process, domain experts and engineers have to assess the most stable and reliable operation states possible for their CPS to reach from almost all preceding operation states. Those would be defined as the safe state envelope, the CPS can operate in most reliably.
Furthermore, engineers have to specify which functionalities to restrict, which processes to suspend, and which communication channels to close, especially in interfaces reaching external networks, when going into degraded operation to fortify the CPS against external threats and failures.
These are both policies that must be embedded in the protective shell prior to operation to enable timely and effective execution of such interventions.

Artificial Intelligence Principles

The protective shell has formulated several requirements regarding the use of artificial intelligence, which are enumerated together subsequently:

Utilise Explainable AI (XAI) One of the key necessities while using AI within the protective shell is to utilise only variants of AI that are intrinsically explainable to humans. This restricts the realm of usable ML models to the simpler models, while the more complex ones often lack explainability on their own and typically need to employ reasoning algorithms to calculate the most likely explanation for their predictions afterwards. This already sheds light on the whole issue of why complex models are not viable for implementation in a protective shell.
One of the main objectives of a protective shell is to remediate negative emergent properties to safeguard the CPS from cascading effects potentially leading to security incidents or safety accidents. However, when the protective shell employs inherently complex functionality that by itself could exhibit such emergent, yet unknown properties, it is not capable of protecting an embraced CPS against it.
Furthermore, it was shown that in the context of protective shells, the accuracy of the employed ML model is lower prioritised than the faithfulness and explainable nature of its predictions. This is of utmost importance during court hearings after safety accidents or security incidents still occurred despite having the protective shell implemented, to prove that the protective shell always acted in the best interest of safeguarding the CPS and can, for every taken action, lay out the exact reasoning behind it without relying on algorithms calculating the most likely explanation. Additionally, the manufacturers are also interested in receiving reliable operation data to further improve their CPS.
ML models having intrinsic interpretability are even more suited for safety and security-critical operation environments since such models can not only provide exact explanations for their predictions but can also lay out the models' behaviour across all scenarios in human understandable terms. This makes it much more practical to evaluate if the current trained model is reflecting the desired behaviour to an acceptable extent, or needs to be further trained.

Employ Supervised and Isolated ML Training Another important point within the AI principles is a guideline for reliably training the ML models for later utilisation in the protective

shell. Since the training approach is not as crucial as other AI characteristics for the protective shell, it should be seen more as a suggestion while arguing why such techniques are seemingly best suited to adapt in CPS scenarios.
First, supervised training techniques are highly suggested, meaning the machine learning model is fed with labelled data records during training, indicating whether the data is from an "operating as intended" state or if it is displaying a state of "abnormal behaviour". This approach is rational, especially for CPS training data, since the operator or domain experts can reliably label the individual data records as one of both categories while observing the system, thereby directly supporting the training process. Semi-supervised training might be also suitable in scenarios where manually labelling the data records requires an unfeasible extra work effort. Regardless of which training process is ultimately chosen, the established model has to be reevaluated by the domain experts and engineers to ensure a precise boundary between truly normal and abnormal behaviour could be established, which directly affects the model's prediction accuracy.
This directly leads to the second guideline for ML model training, which is to clearly separate the training process from the operational use of this model. This is called the isolated training approach, as opposed to applying continuous reinforced learning. An isolated process allows aggregating the training data in a more extensive way by applying feature scaling, outlier elimination, and other procedures aiding during training. Furthermore, it is resilient against adversarial attacks where attackers feed the ML model with specifically crafted data during training to cause them to generate inaccurate predictions. By selecting only faithful data records that best represent the operational conditions the CPS experiences daily, suspicious data gets filtered out before providing it to the training phase to protect the model from such attacks and various other unwanted negative effects. Lastly, the isolated approach is also preferred, especially in safety- and security-critical CPS because the domain experts and engineers have dedicated time to test and verify the resulting model if it represents the desired behaviour as expected.

Minding Performance and Suitability The last crucial point within the AI principles are the performance requirements and the suitability of the model regarding the embraced CPS. The ML model's mechanisms, like regression tasks and classifications, are likely to impose the highest amount of computational effort upon the protective shell. Therefore, it is essential to consider the model's performance in efficiently calculating predictions and classifying the current operating state.
As presented already in this chapter, some parts of the protective shell's control loop and the CPS feedback loop run sequentially. Therefore, it is important that the control loop iterations are at least faster than $\frac{1}{SamplingFrequency}$, the inverse of the sampling frequency, whereas the sampling frequency defines how often the protective shell receives new data records per second, with the ML model optimisation having the biggest influence in achieving this objective. This goes hand in hand with the overall suitability of the model to provide accurate predictions for the embraced CPS. The CPS characteristics, like the dimensionality of the data records, the type of relationships exhibited by the features, or if the training data is prone to creating overfitting models, define the model that is best at dealing with such data.

Manual Overwrite Necessity

Another architecture and design principle is that for every protective shell implementation, the CPS operator must have the opportunity to manually overwrite actions conceptualised by the protective shell or deactivate the protective shell altogether. This is of utmost importance to prevent a tug-of-war scenario where the protective shell and the operator execute competing actions, ultimately leaving the CPS paralysed. Therefore, the human must always

have the upper hand and can force its actions through to the CPS.
The operator's overwrite capability must be achieved through effective human-machine teaming, as laid out in section 7.1.2. Therefore, a decision-making approval pipeline is required while keeping the operator up-to-date with the protective shell detections and interventions through a user-friendly GUI. This GUI also provides the means for the operator to decline or interrupt current intervention plans, as well as adjust intervention plans by overwriting critical actions to represent their own goals.
Furthermore, the operator needs to have the power to deactivate the protective shell if it keeps interfering with the operator's goals, which risks the trustworthy operation of the CPS. This also includes the case where the protective shell on its own experiences software errors during run-time, thus can't safeguard the CPS effectively and reliably. Once the protective shell is deactivated, no functionality by it is provided, and the data streams are directly forwarded to and from the CPS.

Independent CPS Architecture

The protective shell architecture must not impact the underlying CPS architecture; therefore, the CPS remains entirely independent of the protective shell, providing its functionality with or without being embraced by a protective shell.
As a result, the development process of the CPS remains unchanged when the manufacturer decides to implement a protective shell. This is especially advantageous since it does not add complexity to the design-time process for the engineers, and the two implementation procedures can proceed independently, allowing them to be carried out in parallel.
While there are no conceptual changes required for the CPS, the sending and receiving of data streams must be rerouted through the protective shell, which acts as a kind of proxy. Further details on this will be defined in the following principle.
This architectural principle is crucial for achieving broad industry acceptance, since making changes to the sophisticated and well-established CPS development process can be difficult to justify and may be met with scepticism and resistance.

Obligatory Rerouting of Data Streams

This design and architecture principle is essential for the protective shell to be able to provide its safeguarding functionalities. Through the encompassing architecture embracing the CPS, the protective shell completely caps any direct incoming or outgoing connection to the system. However, it provides a rerouting capability by acting like a proxy that receives all data streams and, depending on its predictions, forwards them accordingly to the suitable receiver. Those data streams include, but are not limited to:

- Sensor Readings
- Actuator Reports
- Application Signals
- Network Communication
- Update Requests
- Logs / Operation Data written by the CPS

Therefore, every data stream trying to reach or leave the CPS without being proxied through the protective shell is blocked and discarded. This enables the additional layer of defence in depth characteristics aimed to provide by the protective shell implementation.

Apply Adequate Security Controls

Security controls are immeasurably valuable to the protective shell for being able to safeguard itself and the underlying run-time CPS from malicious threats trying to exploit vulnerabilities. This design principle again summarises what security controls are already present in the protective shell's architecture and suggests additional ones to be implemented, proven suitable in various CPS scenarios, categorised in a framework by the National Institute for Standards and Technology (NIST) [53], especially for CPS and other critical infrastructure.

Fortified Access Control Access control procedures are implemented directly in the CPS, especially when remote access from external networks is required. However, the protective shell supports the access controls with stricter rules by implementing strong identity management for users and devices as well as additional identity and access management capabilities. This allows the protective shell to more consistently block unauthorised or suspicious users by taking additional factors into consideration during authorisation and user action monitoring.

Robust Asset Management Robust asset management is an essential security control and includes the system integrity checks as well as maintaining an inventory of the CPS components.
The system integrity checks encompass a software versioning process, compatibility and completeness validation, verification of full hardware functionality, and the restriction of only executing trusted, signed code.
Furthermore, by maintaining an inventory of the CPS components that marks critical assets (for eventual prioritisation purposes), it enables data stream mapping and enables conducting security assessments during the CPS development lifecycle.

Effective Training and Security Awareness Another security control that should be part of the secure design and architecture principle is effective training of operators handling incidents and the promotion of security risks exhibited by their CPS.
It's important to differentiate between casual operators and professional ones, as their training needs and options can vary significantly. Here, the focus lies on training for professional operators as the manufacturer has limited influence on training casual operators except providing handbooks and documentation.
While the professional operators get certainly trained on the CPS they should control and supervise, this training procedure should be expanded to contain also the protective shell once implemented. There, the future operators must learn the protective shell's capabilities, the GUI, their interactions, and how to deactivate it. Furthermore, common guidelines are suggested on how to react to impending and security incidents in cooperation with the protective shell while creating awareness that trustworthy operation can be affected all the time. For that, typical incident responses and procedures are planned beforehand that can be either fully or semi-automatically executed by the protective shell. Those also include the development of backup and recovery plans during post-incident procedures.
Furthermore, the operators must be trained and made aware of the relevant security standards that the CPS needs to be compliant with at all times.

Sophisticated Process and Network Monitoring Run-time monitoring is one of the key capabilities of the protective shell. Therefore, it proves highly valuable to apply process and network monitoring techniques known from intrusion prevention systems, both network- and host-based, as security controls.

The protective shell should continuously scan for suspicious packets and messages deviating from known patterns that get sent either from one internal component to another or are being sent from an external network to the CPS. Once detected, the protective shell can filter out such packets or messages, thereby proactively preventing eventual intrusion or spreading attempts. Additionally, the packets and messages are not only evaluated upon known patterns but also what payloads they are carrying, for example, by scanning the payload for familiar malware signatures.
There are also process monitoring techniques that should be definitely part of protective shells because they provide essential security controls for early detecting data breaches or successful intrusions within the CPS. Therefore, the protective shell closely monitors processes if they exhibit unusual activities like action patterns correlated to privilege escalation techniques or exhaust large parts of the available system resources.

Enforce Zero-Trust Architecture The zero-trust architecture concept is a valuable addition to the security controls principle since it safeguards the entire communication sent from and to the protective shell while employing continuous authentication efforts and setting strong network segmentations.
Zero-trust architecture emphasises the principle of "never trust, always verify", including that only because an entity is already part of the local network or system, it should not be granted increased trust relative to other outside entities. It operates based on the assumption that threats may already exist within the system or internal network, thereby trying to prevent them from spreading any further.
It includes the guideline to always encrypt data in transit and at rest for all communicating components within the CPS, and that messages being sent must be protected against undetected manipulation with message authentication codes (MACs) or signatures. Furthermore, zero-trust requires continuous authentication for components communicating to verify their identity.
Lastly, strong network segmentation can also be seen as one important capability of zero-trust architectures, hindering malicious threats from moving freely in the internal networks and spreading to devices in different network segments.

In summary of the security controls principle for protective shells, it is important to tailor these controls to the specific requirements and risks of the CPS and its operating environment, as well as regularly update them to address evolving threats and emerging vulnerabilities.

Make use of MAPE-K

Using MAPE-K as the reference architecture for the protective shell is an essential architecture principle presented in this thesis. It suits the concept of the protective shell especially well by providing its control loop, aiding consistent monitoring, analysing CPS components, and planning interventions, which ultimately get executed.
Furthermore, the adoptions discussed for employing human-machine teaming during the MAPE-K loop are highly valuable in the CPS scenario since there are often operators present that the protective shell has to work with cooperatively to achieve the higher objective of trustworthy CPS operation. Each MAPE-K phase has to be elaborated and defined in more detail during the design process of each specific protective shell by adding functionalities like detection mechanisms or interventions from the plugin catalogue introduced in this chapter. This enables the utilisation of MAPE-K as an architecture foundation that can then be adjusted and configured in much detail to purposefully and efficiently address the problems each individual CPS encounters.

7.1.5 Conclusion Architecture

This concludes the generic protective shell architecture section. The MAPE-K reference architecture got adopted and refined to suit the protective shell concept best. There, each individual phase got presented and set in context what part of the functionality and logic of the protective shell is being located there. Furthermore, the whole adjustments that were made to allow a seamless cooperation of the automatic control loop characteristic of MAPE-K and the human-machine teaming procedure got presented, and first suggestions were made on how such teaming could be effectively implemented.
Additionally, a protective shell plugin catalogue got presented, holding recommendations for detection mechanisms that can be employed in the analyse phase and intervention actions that can be utilised during the planning phase. This module library providing crucial advantages like reusability and reliability also enables a very detailed, tailored configuration of protective shells purposefully aimed at safeguarding specific CPS.
Lastly, a whole range of architecture and design principles got listed, retrieved from previous chapters and newly identified during the architecture design in this chapter. The principles provide engineers' guidelines and best practices for developing and implementing their protective shells, thereby increasing industry acceptance for the whole protective shell concept. It transparently lays out blueprints to key characteristics of the protective shell's architecture and provide reasoned suggestions for what is required, mandatory, and obligatory necessities to enable a trustworthy operation.
This marks another pivotal point for the thesis because the second research question can be addressed: "Can a generic core for a protective shell be developed which is applicable to multiple application domains?". By referencing the knowledge base provided by the state-of-the-art, the results from chapter 5, and the architecture and principles introduced in this chapter, a straightforward answer to this question can be provided: Yes, a generic core architecture for a protective shell could be developed and is applicable to multiple application domains.
By adjusting the protective shell's configuration settings according to the CPS specifics, requirements, and engineer preferences, the core architecture forms the basis for developing and implementing protective shells purposefully aimed at safeguarding specific CPSs. This also allows protective shells to be designed to reflect specific behaviours, features, and characteristics of the CPS it safeguards by utilising the plugin catalogue and always minding the presented engineering principles. The principles introduced in this chapter build a solid foundation and a common ground for engineers across various domains to design their protective shells, which can then effectively cooperate with the operators and other components in the CPS environment to enhance trustworthiness.

7.2 Implementation Details

The implementation process for protective shells varies significantly among manufacturers and is highly tailored for the specific CPS for which the protective shell is designed and implemented. This section will provide fundamental considerations without delving into excessive detail about the implementation adjustments.
As mentioned previously, the design, implementation, deployment, and operation of the protective shell should *minimise its impact on the CPS*. It is crucial to avoid any actions that directly interfere with the CPS unless there is a clear and justified purpose. This careful separation between the protective shell and the CPS offers several advantages beyond just industry acceptance.
Firstly, manufacturers can *parallelise* the development processes of both the CPS and the protective shell. Therefore, the time-to-market plans for the CPS should remain unaffected,

while enhancing the trustworthiness of the overall system. This can be achieved by assigning additional engineers to work on the protective shell development while the core CPS development team remains focused on its tasks.
Secondly, it is highly advisable to establish an *independent team of protective shell experts* to develop these protective shells. This approach is beneficial because when two independent engineering teams are working toward the same objective, it becomes unlikely that both teams will implement the same logic flaws, misinterpret requirements, miscalculate risks, or overlook industry best practices. Consequently, the resulting system will have a stronger security and safety posture, reducing vulnerabilities that could otherwise slip through to the final design and architecture. Moreover, the expert team likely already has gathered experience in successfully implementing protective shells, which the project can profit from and expect a high quality implementation.
Furthermore, the protective shell expert team should conduct separate testing to validate their protective shell solution. If vulnerabilities are discovered within the CPS during this testing, they should promptly inform the other team of engineers about these findings. This collaborative approach allows for the mitigation of issues within the CPS before product deliveries starts. *This proactive step is critical to ensure that the responsibility for risk management does not shift from the CPS engineers to the protective shell team*. It ensures that vulnerabilities identified in the CPS design and architecture are addressed before protective shell deployment, regardless of whether the CPS operates with or without the protective shell.
Adhering to this implementation advice safeguards against potential risks manifesting during disorganised development procedures and enhances the overall trustworthiness and reliability of the CPS and its protective shell.

7.3 Evaluation

This section undertakes a comprehensive evaluation of the presented generic protective shell architecture, with a primary focus on assessing the *effectiveness* of employing a protective shell in CPS. This evaluation aims to ascertain whether the introduction of a protective shell introduces new, additional vulnerabilities to the overall system, consequently giving rise to new failure modes and expanding the attack surface. Additionally, in order to make a well-founded hypothesis regarding whether the protective shell ultimately enhances the trustworthiness of CPS operation, it is imperative to examine the extent to which pre-existing failure modes from the original CPS design-time architecture are rectified and how many of the existing attack vectors can be mitigated during run-time.
Lastly, a short summary will be provided, acknowledging the inherent limitations of this evaluation and suggesting it as a foundation for future research, particularly in the practical implementation of protective shells in existing CPS utilising the presented generic framework.

7.3.1 Additional Vulnerabilities introduced by the Protective Shell

It is evident that the implementation of a protective shell increases the overall complexity of the deployed system compared to a standalone CPS. This heightened complexity on its own is one of the driving factors for increasing concerns in recent times regarding the safety and security of modern CPS by experts, resulting to the research problem addressed in this thesis in the first place. The increase in complexity results in emergent properties, which can potentially have negative effects on CPS operation. Furthermore, the introduction of a protective shell substantially increases the system's functionality, as indicated by the expanded code base in comparison to the CPS without a protective shell. Consequently, this expansion provides more opportunities for logic flaws or vulnerabilities to manifest within the final architecture.

Vulnerabilities serve as the root cause of safety accidents and security incidents since they can be exploited by external malicious actors or induce faults during run-time, often resulting in cascading effects across the entire CPS.
One might argue that evaluating the system's trustworthiness should entail a comprehensive assessment of the total vulnerabilities present in the run-time system. However, this perspective falls short in the context of a protective shell scenario. When implementing a protective shell, the total count of vulnerabilities at the very least remains the same, and in most cases, increases significantly in comparison to the number of vulnerabilities present prior to the protective shell's implementation. This occurs because the protective shell does not employ vulnerability elimination or risk management processes to directly reduce the vulnerabilities present in the CPS's design architecture. Consequently, these vulnerabilities remain unchanged, and if vulnerabilities are nested within the protective shell architecture (which is highly likely and difficult to prevent), the overall count increases.
The inadequacy of this approach stems from the fact that the protective shell's primary objective is to prevent vulnerabilities from being exploited or causing faults during run-time by addressing identified failure modes and attack vectors. This does not imply that the protective shell eliminates vulnerabilities entirely; rather, it tries to *prevent them from negatively affecting the CPS during operation*.

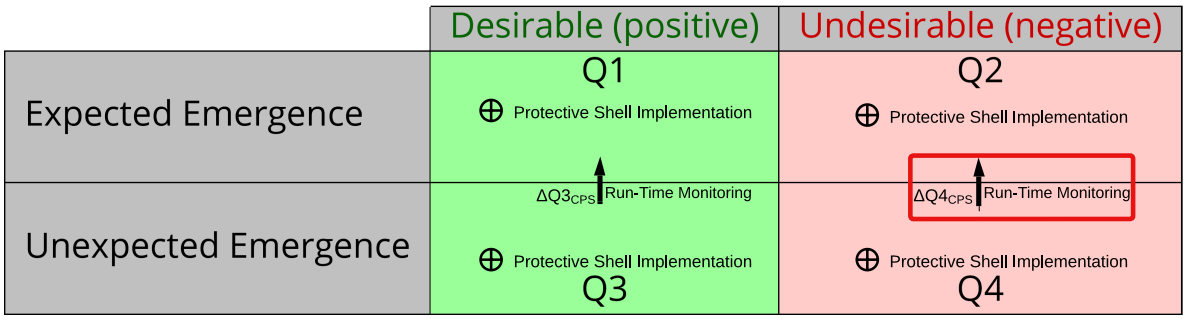


Figure 7.10: Evaluation - Emergence Matrix Protective Shell

This concept can be best visualised by examining the emergence matrix after the protective shell is implemented, as depicted in figure 7.10. Since the embraced CPS is likely a CPSoS either from the start or following the implementation of the protective shell, the emergence matrix is applicable. As indicated by the addition symbols, the implementation of the protective shell results in outcomes being added to every quadrant. However, the most significant developments are observed in the two arrows that shift outcomes from Quadrants 3 and 4 to Quadrants 1 and 2, respectively. These outcomes were initially unexpected in the original CPS but are now expected due to the run-time monitoring by the protective shell. This is particularly crucial for outcomes initially located in Quadrant 4, as they are the primary sources of safety accidents and security incidents.
This transition is possible because the establishment of desired behaviour through functional specifications, rule sets, and, most importantly, behavioural models now defines, and *therefore expects*, the previously unexpected nature of a substantial number of outcomes. This definition is often derived from intrinsic knowledge of logic expressions or within machine learning models, thereby rendering such outcomes as expected cases. It is essential to note that engineers don't need to anticipate these outcomes during the design phase. Instead, when they occur during CPS operation, the protective shell can anticipate and identify them based on its predefined desired behaviour specification, thereby making such outcomes expected.
The set of negative outcomes that were previously unexpected but are now expected due to these new definitions is denoted as $\Delta Q4_{old}$ and shifts to Quadrant 2. This is crucial, as it is only possible to address expected negative behaviour during the development pro-

cess through risk management and vulnerability elimination, or in the case of the protective shell, during run-time through various interventions ranging from corrective and protective actions in real-time, reverting to a safe operation state. Consequently, the first crucial step in the evaluation if the protective shell provides real added value to the CPS trustworthiness is to assess the following equations:

$$Q1 = Q1_{CPS} + Q1_{PS} + \Delta Q3_{CPS}$$

$$Q2 = Q2_{CPS} + Q2_{PS} + \Delta Q4_{CPS}$$

$$Q3 = Q3_{CPS} + Q3_{PS} - \Delta Q3_{CPS}$$

$$Q4 = Q4_{CPS} + Q4_{PS} - \Delta Q4_{CPS}$$

QX_{CPS} - Set of outcomes in QX before PS implementation
QX_{PS} - Set of added outcomes in QX after PS implementation
ΔQX_{CPS} - Set of moved outcomes from QX_{CPS} to $Q(X-2)$ through run-time monitoring

Since Quadrant 4 is the primary focus, the protective shell demonstrates added value in emergent negative properties' identification when it successfully identifies enough unexpected negative emergent properties from the original CPS so that the following condition holds:

$$|Q4| < |Q4_{CPS}|$$

which directly implies:

$$|Q4_{PS}| < |\Delta Q4_{CPS}|$$

Once a portion of the old Quadrant 4 outcomes is successfully identified during run-time by the protective shell and consequently shifted to Quadrant 2, the protective shell aims to provide interventions for every negative outcome that it has moved and for every negative outcome that was already in Quadrant 2 in the original CPS but was either wrongly accepted by the engineers or lacked sufficient mitigation strategies, often due to ignored industry best practices.
If the protective shell can remediate enough Quadrant 2 outcomes so that the combined risks of Quadrants 2 and 4 in the CPS with the protective shell are lower than the combined risks of $Q2_{CPS}$ and $Q4_{CPS}$ in the CPS without the protective shell, it ultimately has a positive influence on the trustworthiness of the CPS by reducing its residual risks.
To achieve this, it is crucial to define effective interventions to address a wide range of possible abnormal behaviours, equipping the protective shell with the tools needed to mitigate as many Quadrant 2 outcomes as possible.
Concluding, this approach provides a pragmatic means of assessing whether the protective shell enhances the trustworthiness of the embraced CPS or introduces more risks than it can mitigate during operation. This evaluation mirrors a practical scenario in which two identical CPS run side by side, with one has a protective shell implemented while the other not, and subsequently determining which of the two CPS operated more securely and safely.

In the next two sections, potential introduction of specific new failure modes and the expansion of the attack surface resulting from the implementation of the protective shell will get analysed. Those are part of the outcomes that get added in Quadrant 2 and 4 depicted in figure 7.10.

New Failure Modes

Some new failure modes may be introduced to the CPS when the protective shell is implemented, as already discussed in chapter 5. While some of these failure modes are specific to the chosen example, some similarities could have been identified.

Firstly, there is the inherent risk that the protective shell may *misinterpret* the current valid operational state as abnormal behaviour. In such cases, interventions may be planned and either directly executed, bypassing operator decision, or the operator may be convinced by the protective shell's explanation and intent, approving the intervention plan that might pose a direct risk to the system's trustworthy operation.
This scenario cannot be entirely eliminated, but two primary strategies can reduce the likelihood and impact of incorrectly identifying abnormal behaviour. The first is to *define the desired behaviour as precisely as possible*, reducing false positives by reevaluating not only functional specifications, rule sets, and policies but also validating the behavioural baseline trained in the machine learning model. Careless training of the model can lead to overfitting, resulting in a higher likelihood of falsely identifying abnormal behaviour.
The second strategy is to *employ a diverse range of protective and corrective actions* in the intervention plan. If the protective shell has a wide variety of options during the planning phase to counteract abnormal behaviour, it can fine-tune the intervention plan in greater detail, reducing the need for rigid and imprudent plans that could overly interfere with the CPS.

Another risk lies in the protective shell's *control loop taking an extended amount of time to complete*, leading to latency issues in the CPS feedback loop and potentially causing operational instability. In scenarios where rapid response times are crucial to prevent impending safety accidents or security incidents, the protective shell may take longer than usual to complete its control loop iterations. This delay is caused by the reevaluation process during the analysis phase, where the protective shell assesses whether the CPS is genuinely deviating from the desired behaviour, coupled with the computational time required to prepare effective intervention plans. Protective shell engineers must be cautious not to become paralysed when timely interventions are essential. Therefore, it is crucial for most CPS not to overly prioritise achieving the utmost certainty in identifying abnormal behaviour or repeatedly adjusting the plan further with the objective to prepare the perfect plan, especially if it consumes an unreasonable amount of time. Time considerations should be an integral part of each protective shell phase and factored into the decision of whether to continue calculating or proceed to the next phase.

Lastly, one potential failure mode concerns *run-time errors occurring within the protective shell itself*. If the protective shell consists of flawed software or encounters unexpected operating environments that lead to failures, it can significantly impact the embraced CPS. Since these failures cannot be fully addressed as discussed in section 4.1, adherence to industry best practices in design, engineering, and architecture principles becomes essential to enhance the reliability of the protective shell application. In cases of failures, operators must be trained on how to deactivate the implemented protective shell promptly, in the hopes of doing so before cascading malfunctions spread throughout the CPS, potentially resulting in safety accidents.

Enlarged Attack Surface

A few new attack vectors might emerge in the CPS following the implementation of the protective shell, as discussed in chapter 5. While some of these attack vectors are specific to the chosen example, similarities have been identified.

Attacker might try to *manipulate the protective shell* in intervening to their attacks by executing harmful interventions at which they aim for. For instance, an attacker may simulate a brute-force attack by rapidly iterating through username and password combinations mimicking to try to gain access to the CPS. The protective shell swiftly identifies this unusual authentication message pattern and initiates intervention plans. One such intervention could be the rate limiting of login attempts uniformly for all users. While this significantly reduces the effectiveness of brute-force attacks, it may also unintentionally block legitimate users from authenticating when the attackers use the rate limited bandwidth completely for them, leading to a successful denial-of-service attack.

Furthermore, the *protective shell presents a significant risk to the CPS once it is compromised* due to its supervisory functionality and elevated permissions. If attackers discover vulnerabilities to exploit within the protective shell, enabling successful infiltration and control, a CPS-wide security breach is unlikely to be stopped. Additionally, since protective shells developed within a single manufacturer may share similarities, a discovered vulnerability could potentially be applicable to the entire CPS product catalogue of that manufacturer, resulting in widespread exploitations across multiple CPS instances. The same risk applies to the manufacturer's certificate used for signing valid update requests sent to the protective shells. If this certificate is stolen, as is increasingly common, attackers could potentially infect all CPS instances with protective shells that trust this certificate by crafting malicious update requests.

One last attack vector the protective shell exhibits is related to the *direct data stream connection to the actuators and sensors*. These communication channels are often inadequately secured with cryptography providing no confidentiality and integrity. However, if encryption and integrity protection, as recommended in the design principles, were implemented, this attack vector would be mitigated. Nevertheless, due to performance considerations and the typically wired nature of these connections, communication protection is often neglected. As a result, attackers could easily manipulate application signals sent to actuators or sensor readings reported to the protective shell. This manipulation could be used for reconnaissance purposes to understand the inner workings of the CPS or to induce instability in the CPS, prompting the protective shell to intervene. Similar to the first attack vector mentioned in this section, attackers could attempt to manipulate the protective shell into initiating protective shell interventions on their behalf.

7.3.2 Summary

This evaluation has provided a quantitative approach for assessing the effectiveness of protective shells in CPS. While this remains theoretical within this thesis, it serves as a foundation for future work that would involve the practical implementation of protective shells using the generic framework presented here, while also adhering to the specified architectural and design principles, to evaluate the architecture in more detail.
While the generic protective shell evaluation lacked a practical approach, the specific protective shells developed in this thesis have demonstrated effective solutions to the challenges faced by CPS. Therefore, it is reasonable to suggest that with the adoption of the presented MAPE-K reference architecture, its configuration through chosen plugins, and adherence to

the specified principles as guidelines, the generic protective shell has the potential to improve the trustworthy operation of safety- and security-critical CPS.
Therefore, an outlook has been provided regarding the potential introduction of failure modes and attack vectors when implementing a protective shell. It's evident that neither the count of attack vectors nor the number of failure modes decreases with the mere implementation of a protective shell; rather, *they tend to increase due to increased complexity*. Therefore, it's essential to highlight that *the effectiveness of the protective shell becomes apparent only when, during run-time, a greater number of emergent negative behaviours, combined with pre-existing negative outcomes in Quadrant 2, can be effectively mitigated compared to what the original CPS design would have experienced*.
It is important to note that trustworthiness, while a primary focus of this thesis, is not the sole criterion manufacturers consider. Profit-oriented companies often assess the implementation overhead and evaluate it based on a price-to-performance ratio. Performance, in this context, typically refers to the damage to a brand's reputation that may result from safety accidents or security incidents, and the associated legal liabilities and compensation payments. This assessment varies across different industries; for example, manufacturers in the aviation industry may face significantly higher brand damage and compensation costs following a safety accident compared to those in the agriculture industry employing automated harvesters.

Furthermore, this marks a suitable point to address the third research question: "Is it possible to mitigate the risks that emerge when implementing a protective shell to reduce overall residual risks for safety and security?".
This question cannot be definitively answered within this thesis, as it requires practical case studies implementing protective shells according to the proposed architecture. These case studies would yield empirical data necessary for a more precise answer. However, even without the benefit of extensive practical data, this thesis provides a partial response by examining the specific protective shells developed. In these cases, *the mitigated failure modes and attack vectors consistently exceeded the newly introduced vulnerabilities*. The evaluation method presented in this section suggests that, indeed, the residual risks for CPS are reduced after the implementation of a protective shell compared to before its implementation.
Moreover, this thesis stated the hypothesis that by rigorously following the design and architecture principles discussed in this chapter, adopting the MAPE-K architecture, and configuring the protective shell with careful consideration of detection mechanisms and intervention actions, as well as defining desired behaviour clearly, the protective shell could contribute to reducing overall risks for safety and security in CPS across various application domains.
Therefore, the answer to the third research question is "more likely than not", pending further empirical validation through practical implementations and assessments.

8 Conclusion

The conclusion begins with a concise summary section, providing a precise recapitulation of the key points discussed in the previous chapters. Subsequently, the research questions will be examined in greater depth, offering well-reasoned and substantiated answers, along with the relevant context and the original sections where these questions were addressed. Moving forward, an overview of this thesis' contributions will be presented, spanning from its role as an introductory tutorial for readers to the refinement of the generic protective shell concept, thereby contributing to the scientific knowledge base.
Following, open points that require further exploration in future work are highlighted, outlining potential directions for ongoing research.
Concluding this study, the two most important recommendations are recapitulated from the thesis, holding the core insights.

8.1 Summary

The *design science research methodology* outlined in chapter 2 proved highly effective throughout the thesis. It offered a solid structure, beginning with the identification of the CPS environment, its application domain, actors, organisation, challenges, and opportunities. The relevance cycle provided real-world insights into existing infrastructure, processes, capabilities, and expectations towards the design science research.
The research problem was quickly identified and articulated as the increasingly difficult venture of designing safe and secure complex CPS solely during the development process. The potential consequences of not finding a solution were articulated: either *limiting CPS complexity* to ensure the resulting design architecture to be comprehensible by the engineers, thereby reducing the operational risks to an acceptable level, or *pursuing higher complexity while risking untrustworthy CPS operation*, thus increasing the likelihood for catastrophic outcomes.

As complex CPS offer substantial benefits across various sectors, the pressure on the design science research to address this research problem is significant. The existing scientific knowledge base offered concepts, methods, and theories as a foundation. The rigor cycle described the state-of-the-art about CPS in chapter 3 and the fundamental protective shell concept in chapter 4.
Within the CPS chapter it was shed light on their *safety- and security-critical aspects* by directly interacting with the physical world, the *engineering principles* defined to guide the development process and the modern strive to connect constituent systems together to a *system-of-systems* that can achieve higher objectives. This increased complexity by designing CPSoS

was explored, highlighting emergent properties that might have a negative impact on the system and the manifested concealed vulnerabilities in the design architecture. Therefore, the *safe and secure operation during all operating conditions can't be ensured any more*, like it could be for simpler CPS.
In the subsequent chapter, still contributing to the state-of-the-art, the protective shell concept was introduced, offering a promising solution by enhancing the CPS run-time architecture. The run-time monitoring paradigm was presented, suggesting enhanced trustworthy CPS operation. The expectations and goals for the protective shell were formulated. However, it is important to note that this concept remained theoretical at this stage, lacking an in-depth analysis of its viability to address the research problem.

Therefore, chapter 5 took a practical approach with case studies to assess if protective shells could mitigate undesired behaviour and concealed vulnerabilities. Well-documented safety accidents and security incidents involving CPS were studied, by retrieving the inherent vulnerabilities and developing specific protective shells. During the overall evaluation, the protective shell similarities across the chosen examples were identified, revealing a *generic core and characteristics applicable to all developed protective shells*. Furthermore, the evaluation demonstrated for each case study increased operation trustworthiness, even when the protective shell introduced new vulnerabilities to the overall system.

Nevertheless, the approach from this chapter 5 was a *theoretical academic scenario* supported through the underlying deterministic knowledge from the post-incident / accident reports that helped identify the vulnerabilities through a retrospective view. This would obviously not help to mitigate current vulnerabilities that never surfaced before, like zero-day exploits. Therefore, with the following chapter 6 about artificial intelligence, there is made an attempt to bridge from a retrospective view on safeguarding CPS to a present view. This enables the protective shell to mitigate some of the deeply hidden vulnerabilities or negative emergent properties when they occur for the first time, that have not yet caused a malfunction or were not yet exploited with malicious intent.
That is why the protective shell employs *advanced anomaly detection* that relies on the utilisation of machine learning models. The benefits of AI were explored, and guidelines for using explainable and interpretable models were established. Lastly, suitable machine learning models were presented, demonstrating their potential for accurate anomaly detection and their more flexible character when comparing to the rather static functional specifications.

With the state-of-the-art basic concepts, the common ground derived from the different protective shells and the advanced anomaly detection capabilities, the last chapter 7 combines everything to a *generic protective shell framework*.
The MAPE-K reference architecture was found to be suitable for the protective shell concept and got further refined. First, for better computational efficiency, two asynchronous jobs were incorporated, one for the analysing regression tasks and the other for preparing intervention plans for likely abnormal behaviour. Furthermore, addition transitions got added, together with new decision-making steps, and *human-machine teaming* capabilities. Additionally, a *plugin catalogue* offered a wide variety of detection mechanisms and intervention actions for more detailed customisation.
Furthermore, all *design and architecture principles* found throughout the thesis got enumerating. They range from necessary security controls to mandatory manual overwriting and previously mentioned AI guidelines.
Concluding the protective shell architecture, it got shown that a generic framework could be developed applicable to multiple application domains.

The final part of the chapter was dedicated to specifying some implementation details and a pragmatic evaluation of the presented generic protective shell architecture. It got shown that when the *protective shell does not match the characteristic of the embraced CPS* (is not precisely configured), numerous new failure modes appear and attackers find an enlarged attack surface. While highlighting those common drawbacks, the individual engineers can proactively assess if it affects their implementation project and implement counteractive measures.
Therefore, the chapter emphasised the importance of precise configuration for efficiency, accuracy and an overall improvement of trustworthy operation.

8.2 Research Questions Evaluation

In this section, the research results regarding the research question are concisely summarised.

Can the trustworthiness of safety and security-critical cyber-physical systems be improved by implementing a protective shell?

The first research question tackled a fundamental exploration revolving around the overall applicability of the protective shell concept in addressing the research problem. This question is not concerned about how suitable or effective the protective shell addresses the research problem; rather, it questions whether the implementation of protective shells can enhance the trustworthy operation of individual CPS *at all*.
This question was answered in chapter 5.8.3, following the development and evaluation of a representative set of specific protective shells designed to mitigate safety accidents and security incidents. The answer provided in this chapter was a definitive ”yes“. The argument put forth was that across all specific protective shell implementations, the overall trustworthiness of the systems was improved by addressing previously existing failure modes and attack vectors. Furthermore, it was demonstrated that the protective shell concept encompasses capabilities that facilitate a wide variety of mitigation mechanisms. For each unique protective shell, suitable mitigation mechanisms could be identified and employed, thereby addressing vulnerabilities present in the design-time architecture of the CPS or significantly reducing the potential damage caused by accidents and incidents.
Given that the research question was specifically formulated to require evidence from just a single example rather than holding true for all instances, the question can be confidently answered. With 7 out of 7 case studies demonstrating the enhancement of CPS trustworthiness through protective shell implementation, it can be concluded that there is the possibility to improve safety and security during run-time.

Can a generic core for a protective shell be developed which is applicable to multiple application domains?

The second research question aimed to ascertain the universally applicable nature of the protective shell, which is crucial for achieving widespread industry acceptance. Even if the first research question is answered with yes, it does not entirely address the research problem faced by stakeholders in the CPS application domain. This question holds particular significance in determining whether the protective shell concept can *genuinely assist manufacturers with a diverse range of CPS in enhancing their trustworthiness* on a broader scale.
Early indications toward an answer to this question were provided in chapter 5, where lessons learned were summarised, and the generic core was retrieved. This core elucidated the primary protective shell characteristics that became evident as early as section

5.8.3. The specific protective shell similarities got taken over to chapter 7 where the generic framework got introduced.
The framework's foundation was rooted in the MAPE-K reference architecture, which was further adapted to fit the protective shell's requirements. This adaptation introduced two asynchronous processes and new transitions to streamline processes of the reference architecture template. Additionally, the concept of human-machine teaming was implemented, fostering cooperation between the protective shell and human operators.
Furthermore, the framework incorporated a protective shell plugin catalogue encompassing diverse detection mechanisms and interventions based on protective or corrective actions. *This modular approach rendered the generic framework highly configurable for various application scenarios* and supports the concurrent development effort in providing new modules later on. Lastly, design and architecture principles were enumerated to guide engineers through the design, implementation, deployment, and maintenance of specific protective shells. These principles provided standardised processes, highlighted potential challenges, and helped prevent obvious design flaws.
With these contributions in place, the definitive answer to the research question was presented during the conclusion of the generic protective shell architecture in section 7.1.5, confidently answering "yes".

Following the development and evaluation of the seven specific protective shells, along with the identification of their similarities, an initial indication of a generic core applicable across multiple application domains emerged. Notably, the chosen safety accidents and security incidents involving CPS were deliberately selected to encompass a wide variety of industry sectors. Even when analysing systems from sectors such as aviation, healthcare, research, automotive, and critical infrastructure, unmistakable similarities could be retrieved. Furthermore, the adoption of machine learning models for advanced anomaly detection further supported the notion of universally applicability. *These models can be trained using operational data and log files from individual CPS, making them highly adaptable for specific scenarios*.
To conclude, the second research question could be answered to a satisfying extent.

Is it possible to mitigate the risks that emerge when implementing a protective shell to reduce overall residual risks for safety and security?
The third and final research question confronted in this thesis delves into the potential risks that arise when implementing a protective shell. The question tries to assess whether the additional vulnerabilities that the protective shell introduces increase the residual risk, or if the protective shell employed mitigation mechanisms are rectifying enough risks that the overall residual risk is shrinking when implementing it.
This research question is more sophisticated since it already hints to the fact that the protective shell does not only contribute towards a more trustworthy operating CPS. As stated previously, the protective shell is only as effective as it fits the embraced CPS. In a theoretical scenario when it matches completely, thereby having the perfect desired behaviour defined and the most effective interventions employed, then the protective shell does not introduce any additional failure modes or attack vectors. However, such precise configuration is not achievable on a consistent basis. That is why the question is legitimate, even if it is at first glance counter-intuitive, that a protective shell can have opposing effects, when the implementation drawbacks result in higher operation risks.
Again, the specific protective shell chapter 5 gives the first clues, because it was observed that the *remediation of existing failure modes and attack vectors outweighed the introduction of new vulnerabilities consistently across all case studies*. However, since the question is concerned about the general reduction of residual risks after protective shell implementations,

a more in-depth exploration is required, focusing on the generic protective shell instead of the specific protective shells developed.
Therefore, the generic protective shell chapter contained an evaluation section. There a pragmatic evaluation methodology got presented hinting to the emergence matrix outcome changes once the protective shell got implemented. There it became evident that the desired behaviour definition within the protective shell is crucial in *transferring the unexpected, negative outcomes from quadrant 4 to the expected, negative outcomes in quadrant 2*. Once these outcomes got transferred, the second important step in mitigating negative emergent properties, failure modes and malicious threats is to initiate appropriate interventions. Therefore, it is key to have a wide variety of protective and corrective actions to prepare suitable and highly adaptive intervention plans.
To conclude, the precise configuration of the specific protective shell is of utmost importance and decisive if ultimately the protective shell reduces the residual operation risk of the CPS, or in total adds additional risks.
To support engineers to properly configure their protective shell a multitude of components aiding this process were presented, like various mitigation mechanism, anomaly detections (including ML models), interventions, HMT with GUI variances as well as architecture and design principles. These components either contributed by guiding the engineers through the protective shell development making care that no obvious weaknesses would be implemented from the base architecture, or they provided options to configure the protective shell in detail, making it more suitable for the CPS scenario.
Nevertheless, with that in mind but without practical examples implementing the generic protective shell within an actual CPS, the answer to the third research question was ”more likely than not“ and was given in chapter 7.3.2 evaluating the generic protective shell. This answer is sufficient because it considers the whole range trade-offs when implementing protective shells and compares it with the reasoned benefits of CPS on a theoretical basis. However, it leaves room for a more *profound evaluation addressing this research question*.

8.3 Contribution

This thesis has made significant contributions to the protective shell research regarding safe and secure operation of CPS. While some of the contributions have been discussed in previous sections, this section shortly summarises and structures them.

Introductory Tutorial

The thesis provided the reader an *introductory tutorial* on essential topics, enabling a foundational understanding of the research context:

Design Science Research Methodology: A clear explanation of the design science research methodology was presented. The cycles and pillars of this methodology are introduced, demonstrating how they collaboratively contribute to a *problem-oriented research approach*.

Cyber-Physical Systems: An in-depth exploration of CPS is provided, highlighting their tremendous benefits, challenges, and critical aspects related to safety and security due to their direct interaction with the physical world. Furthermore, the scientific definition of risk is outlined, along with methods for applying risk mitigation techniques in the CPS context, emphasising adherence to engineering principles. Lastly, the effects of CPSoS were shown highlighting their enhanced functionality, but also hinting to their increased complexity and the according challenges emerging from this.

Protective Shell Concept: The thesis introduced the reader to the protective shell concept, outlining its potential as a promising solution for the research problem. Moreover, the protective shell architecture got presented, showing it employing run-time monitoring and the resulting capabilities. Additionally, it sets forth expectations for the protective shell that were previously unfulfilled.

Artificial Intelligence and Machine Learning Models: An informative tutorial in the artificial intelligence chapter provides readers with fundamental concepts related to machine learning models, training approaches, and their inherent characteristics. Various use cases, suitable for the protective shell context, were explored. The chapter further introduced specific machine learning models suitable in the protective shell context and delved into their training processes and advanced anomaly detection capabilities.

These introductory tutorials consolidate crucial knowledge for the reader, which were based on *extensive scientific literature review* of the current-state-of-the-art. By synthesising and delivering condensed insights into these critical aspects, the thesis empowers readers to grasp the foundations before delving into more complex discussions.

Safety Accidents and Security Incidents Case Studies

This section contains two separate contributions, which are combined for better clarity.
Besides the short introduction of the case studies being similar to an introductory tutorial, the main contribution was the post-accident or post-incident *forensic analysis of the chosen catastrophic examples involving CPS*. This was done to retrieve the system's architecture vulnerabilities and weaknesses. Importantly, the analysis was conducted only from reports published by trusted sources to retain scientific meaningfulness. After the vulnerabilities and weaknesses for seven familiar safety accidents and security incidents from various industry sectors got retrieved, the second part of this contribution starts.
For each individual example, *one specific protective shell got developed*, equipped with purposefully employed mitigation mechanisms that, in the best case scenario, remediate identified failure modes and attack vectors. Once the specialised mitigation mechanisms got presented, an evaluation per specific protective shell is conducted to assess the effectiveness in rectifying the hazardous situation.
Lastly, the similarities between the mitigation mechanisms employed across different protective shells and their evaluation got retrieved and enumerated. This derived common ground is the most important contribution of the conducted case studies as it hints to a generic protective shell core, their inherent inner workings and the main characteristic, which is highly valuable for the subsequent contribution.

Generic Protective Shell Architecture

The generic protective shell architecture is the primary contribution of this research as all previous efforts are combined and specifically absolved to establish this architecture framework, therefore working towards this higher objective.
There the generic core and characteristics from chapter 5 as well as the experience in developing and evaluation specific protective shells gets utilised. Furthermore, the state-of-the-art for CPS, the protective shell concept and machine learning models get employed to form the new generic protective shell artefact.
Within this contribution, the MAPE-K reference architecture got adopted by adding asynchronous processes for more efficiency and new transitions from phases that were in the original architecture not possible. Additionally, the human machine teaming paradigm got implemented to enable cooperation between the CPS operator and the protective shell.
The detection mechanisms and intervention actions as presented in previous chapters are

being included into the plugin catalogue, enabling modularity for highly customisable specific protective shells. Moreover, the design and architecture principles as well as the implementation details give the engineers guidance in how to develop, implement, maintain and deploy their protective shell.
Lastly, a fundamental evaluation method is specified to verify if the implemented protective shell does fulfil its objective in enhancing the embraced CPS's trustworthy operation.
With the architecture laid out, designed to be universally applicable in various application domains, the thesis contributes to a possible increase in industry acceptance for run-time monitoring and real-time intervention systems such as the protective shell, which leads directly to the next and last contribution.

Enriching the Knowledge Base
Since this thesis utilises the design science research methodology, it is essential to *contribute the successful evaluated artefacts back to the scientific knowledge base*, allowing subsequent research efforts. The fundamental artefact that the knowledge base will be enriched with is the generic protective shell architecture, allowing fellow scientist to adopt, refine and extend the laid out generic concept. However, since the design cycle was iterated several times within this thesis, there are many more artefacts that might aid future research within this field of study.
This thesis could answer the two of the three imposed research question completely and one partially, thereby effectively filling identified knowledge gaps. Furthermore, the thesis could successfully help in solving the research problem identified at the beginning. Since the proposed artefact is not sophisticated and refined enough to impose a solution on its own, it nonetheless helped in making a step in the right direction towards addressing the problem.
Fortunately, as this thesis has no restriction in the form of a blocking note, it can be made public immediately after publishing.

8.4 Future Work

There are several points of interest regarding future work, as previously mentioned. First and foremost, it would be highly valuable if the generic protective shell architecture would get *empirically validated through practical implementations and assessments on actual CPS*. This would without doubt identify several improvements to be made to this thesis' suggestions.

Furthermore, the collaboration between the CPS operator and the protective shell emerged as an important factor during the course of this study. During the development of the specific protective shell, the necessity for meaningful alerts and extensive log files became evident. Consequently, the implementation of an interface enabling communication between the protective shell and the operator appeared necessary. Through the deeper elaboration of human machine teaming it turned out that not only the protective shell must have a means to communicate with the operator but also vice versa. Therefore, it is indispensable to *address the shortcomings of the proposed GUI in future research*, for example through focus group interviews, as its current usability falls short. This would also encompass the creation of at least a second GUI template for the casual user, that has entirely different objectives when working with the protective shell than the professional user.

Additionally, the section discussing artificial intelligence and machine learning models mostly served as an introductory tutorial, lacking the appropriate depth required for seamless integration into the generic protective shell architecture. Therefore, directing research

efforts towards this domain is essential, as it enables to a large part the advanced anomaly detection, protecting the CPS against unexpected, negative behaviour. Particularly, *enhancing the explainability of more complex models is a current area of active research*, promising higher model accuracy and efficiency.

Lastly, the interventions presented, including protective and corrective actions, would benefit from more variety and applicability. Although this thesis formulated an algorithm to identify error sources and the plugin catalogue contained some effective interventions, there remains substantial room for improvement. As indicated by the evaluation, a diverse range of interventions enhances the suitability of the prepared plan in addressing identified abnormal behaviour.

8.5 Recommendation

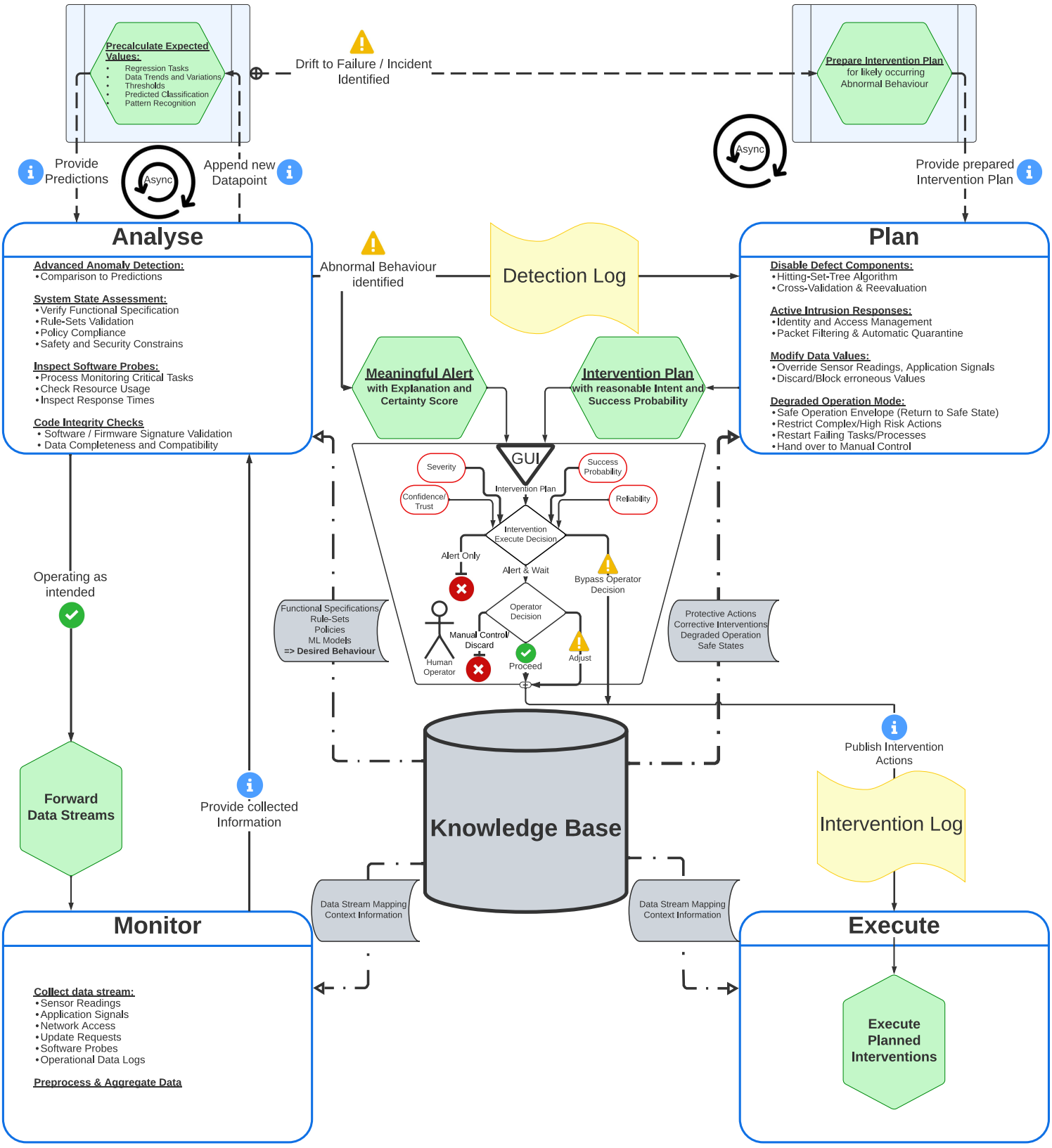
To keep this section concise and avoid repetition, almost all recommendations are enumerated in the design and architecture principles for the generic protective shell 7.1.4. The following two key points offer a summarised overview without delving into too much detail and should be seen as the most fundamental recommendations, when considering implementing a protective shell.

First and foremost, the protective shell is a concept to safeguard complex CPS during operation. That means that simple CPS, where the engineers have a sound overview about the overall inherent behaviour and can to a large part rule out significant design flaws, are *not recommended implementing a protective shell*. The protective shell introduces certain drawbacks, as discussed in Section 7.3.1, including latency due to data stream collection, aggregation, and analysis. When abnormal behaviour gets identified, this latency grows even larger. Simpler CPS often possess the capabilities to effectively address potential operational issues or changes independently, as they tend to exhibit fewer unexpected and negative emergent behaviours and normally don't have highly concealed vulnerabilities.

Conversely, in safety- and security-critical domains involving complex CPS, *implementing a protective shell to mitigate run-time risk is strongly recommended*. Manufacturers aiming to realise the tremendous benefits of the most advanced and complex CPS while struggling to adequately manage operational risks solely during the development process can find the protective shell a viable solution. It's imperative to configure each protective shell meticulously to align with the specific characteristics of the CPS it encompasses, as a mismatch could lead to adverse effects. Accepting elevated residual risks outside the norm to capitalise on the complex CPS benefits is not recommended, as history shows hinting to the analysed safety accidents and security incidents, operating with unacceptable residual risks can have catastrophic outcomes.
These primary recommendations encapsulate the essence of this thesis. For a more in-depth view upon all recommendations made throughout this thesis, delving into the design and architecture principles in section 7.1.4 is strongly recommended.

With this, the exciting study about addressing emergent safety accidents and security incidents in CPS with a protective shell concludes. This thesis not only advances the comprehension of a generic runtime monitoring and real-time intervention concept to mitigate negative emergent properties but also establishes a foundation for future research in the previously mentioned areas. Hopefully, these research efforts contribute to broader industry acceptance of protective shell-like concepts, thereby ensuring the reliable operation of complex CPS.

Appendix



Appendix - MAPE-K Protective Shell Architecture Overview Detail

Statement of authorship

I hereby certify that I have authored this document entitled *Mitigating Emergent Safety and Security Incidents of CPS by a Protective Shell* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. During the preparation of this document I was only supported by the following persons:

Prof. Dr. Frank J. Furrer
Dr.-Ing. Sebastian Götz

Additional persons were not involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, 4th September 2023

Leonard Wagner

References

[1] L. Eliot, *AI Guardian Angel Bots for Deep AI Trustworthiness: Practical Advances in Artificial Intelligence (AI) and Machine Learning*. LBE Press Publishing, Oct. 24, 2016, ISBN: 978-0-6928-0061-4.

[2] F. J. Furrer, "Safe and secure system architectures for cyber-physical systems," Apr. 2023. [Online]. Available: https://link.springer.com/article/10.1007/s00287-023-01533-z.

[3] A. Hevner and S. Chatterjee, *Design Research in Information Systems*. Springer New York, NY, May 14, 2010, ISBN: 978-1-4419-5652-1.

[4] A. B. Brendel, T.-B. Lembcke, J. Muntermann, and L. M. Kolbe, "Toward replication study types for design science research," *Journal of Information Technology*, vol. 36, no. 3, 2021. [Online]. Available: https://doi.org/10.1177/02683962211006429.

[5] A. Romanosky and F. Ishikawa, *Trustworthy Cyber-Physical Systems Engineering*. FL USA: CRC Press, 2017, ISBN: 978-1-4987-4245-0.

[6] R. Alur, *Principles of Cyber-Physical Systems*. MIT Press Ltd, Apr. 10, 2015, ISBN: 978-0-262-02911-7.

[7] F. J. Furrer, *Safety and Security of Cyber-Physical Systems*. Springer Fachmedien Wiesbaden, 2022, ISBN: 978-3-658-37181-4.

[8] D. E. Geisberger, D. M. V. Cengarle, P. Keil, J. Niehaus, D. C. Thiel, and H.-J. Thönnißen-Fries, "Cyber-physical systems - driving force for innovation in mobility, health, energy and production," *acatech - National Academy of Science and Engineering*, Nov. 2011. [Online]. Available: https://www.acatech.de/wp-content/uploads/2018/03/acatech_POSITION_CPS_Englisch_WEB-1.pdf.

[9] C. Cheng, J. Fu, H. Su, and L. Ren, "Recent advancements in agriculture robots: Benefits and challenges," *MDPI - Machines*, Jan. 1, 2023. [Online]. Available: https://doi.org/10.3390/machines11010048.

[10] Tesla, "Tesla vehicle safety report," Jan. 2023. [Online]. Available: https://www.tesla.com/VehicleSafetyReport (visited on 06/26/2023).

[11] B. Moye, "Aaa: Fear of self-driving cars on the rise," Mar. 2, 2023. [Online]. Available: https://newsroom.aaa.com/2023/03/aaa-fear-of-self-driving-cars-on-the-rise/ (visited on 06/26/2023).

[12] A. for Highway & Auto Safety, "Public opinion polls show deep skepticism about autonomous vehicles," Jul. 22, 2019. [Online]. Available: https://saferoads.org/wp-content/uploads/2020/01/AV-Public-Opinion-Polls-7-22-19.pdf (visited on 06/26/2023).

[13] G. Kavallieratos, S. Katsikas, and V. Gkioulos, "Cybersecurity and safety co-engineering of cyberphysical systems - a comprehensive survey," *Future Internet*, Apr. 2020. DOI: 10.3390/fi12040065. [Online]. Available: https://www.mdpi.com/1999-5903/12/4/65.

[14] "Residual risk," *National Institute of Standards and Technology*, [Online]. Available: https://csrc.nist.gov/glossary/term/residual_risk.

[15] P. A. Networks, "What is a zero trust architecture," 2023. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture (visited on 07/15/2023).

[16] F. A. Administration, "Summary of the faa's review of the boeing 737 max," Nov. 18, 2020. [Online]. Available: https://www.faa.gov/foia/electronic_reading_room/boeing_reading_room/media/737_RTS_Summary.pdf (visited on 05/23/2023).

[17] [Online]. Available: https://code7700.com/images/aero/climb_gradient_to_angle.jpg (visited on 08/10/2023).

[18] C. Simon, *Deep Learning and XAI Techniques for Anomaly Detection*, 1st ed. Packt Publishing Ltd., Jan. 1, 2023, ISBN: 978-1-80461-775-5.

[19] W. contributors, "Therac-25," Wikipedia, Ed., Jun. 14, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Therac-25 (visited on 06/22/2023).

[20] N. G. Leveson and C. S. Turner, "An investigation of the therac-25 accidents," I. of Electrical and E. E. (IEEE), Eds., Jul. 1993. [Online]. Available: https://www.cs.columbia.edu/~junfeng/08fa-e6998/sched/readings/therac25.pdf (visited on 06/22/2023).

[21] S. Baase, *A gift of fire : social, legal, and ethical issues for computing and the Internet*. Upper Saddle River, NJ, 1996, Chapter 8.2 Case Study, pages 425ff, ISBN: 978-0-1360-0848-4.

[22] T. Gneuß, "Assessment of risk and danger-potential of stuxnet-like malware on critical infrastructure," *Hauptseminar (Prof. Dr. Frank J. Furrer)*, Jul. 1, 2022.

[23] W. contributors, "Stuxnet," Wikipedia, Ed., Jun. 20, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Stuxnet.

[24] P. Mueller and B. Yadegari, "The stuxnet worm," U. of Arizona, Ed., 2012. [Online]. Available: https://www2.cs.arizona.edu/~collberg/Teaching/466-566/2012/Resources/presentations/topic9-final/report.pdf.

[25] D. of Transportation. "Nasa's toyota study released by dept. of transportation." NASA, Ed. (Feb. 8, 2011), [Online]. Available: https://www.nasa.gov/topics/nasalife/features/nesc-toyota-study.html.

[26] B. Allen. "Toyota sudden unintended acceleration lawsuit ends in landmark verdict." (Nov. 5, 2013), [Online]. Available: https://www.beasleyallen.com/article/toyota-sudden-unintended-acceleration-lawsuit-ends-in-landmark-verdict-2/.

[27] D. C. Miller and C. Valasek, "Remote exploitation of anunaltered passenger vehicle," Aug. 10, 2015. [Online]. Available: https://illmatics.com/Remote%20Car%20Hacking.pdf (visited on 06/06/2023).

[28] P. Bock, J.-P. Hauet, R. Foley, and R. Françoise, "Ukrainian power grids cyberattack," I. S. of Automation, Ed., Apr. 2017. [Online]. Available: https://www.isa.org/intech-home/2017/march-april/features/ukrainian-power-grids-cyberattack (visited on 06/17/2023).

[29] U. H. S. NCCIC, "Cyber-attack against ukrainian critical infrastructure," Mar. 7, 2016. [Online]. Available: https://legacy-assets.eenews.net/open_files/assets/2016/07/19/document_ew_02.pdf (visited on 06/17/2023).

[30] T. Hepher, "Airbus knew of software vulnerability before a400m crash," Reuters, Ed., Nov. 8, 2017. [Online]. Available: https://www.reuters.com/article/us-airbus-a400m/airbus-knew-of-software-vulnerability-before-a400m-crash-idUSKBN1D819P (visited on 06/19/2023).

[31] T. Hepher, "Exclusive: A400m probe focuses on impact of accidental data wipe," Reuters, Ed., Jun. 9, 2015. [Online]. Available: https://www.reuters.com/article/us-airbus-a400m-idUSKBN0OP2AS20150609 (visited on 06/19/2023).

[32] C. P. de Silva, "Judge drops criminal case on a400m crash but points to airbus and epi," E. Confidencial, Ed., Apr. 6, 2018. [Online]. Available: https://www.defense-aerospace.com/judge-drops-criminal-case-on-a400m-crash-but-points-to-airbus-and-epi/ (visited on 06/19/2023).

[33] V. R. Caivano, "Germany and france delay decision on airbus military transport," T. N. Y. Times, Ed., Jun. 11, 2009. [Online]. Available: https://www.nytimes.com/2009/06/12/business/global/12airbus.html?_r=1&ref=europe (visited on 06/19/2023).

[34] "Artificial intelligence act," *European Parliamentary Research Service*, Apr. 2021. [Online]. Available: https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf.

[35] X. Huang, G. Jin, and W. Ruan, *Machine Learning Safety*, 1st ed. Springer Nature Singapore, 2023, ISBN: 978-9-811-96814-3.

[36] [Online]. Available: https://people.duke.edu/~rnau/411trend_files/image004.png (visited on 08/06/2023).

[37] [Online]. Available: https://miro.medium.com/v2/resize:fit:681/0*OYKhBzjmyvnjkgiU.png (visited on 08/06/2023).

[38] [Online]. Available: https://www.youtube.com/watch?v=ZVR2Way4nwQ (visited on 08/07/2023).

[39] [Online]. Available: https://scikit-learn.org/stable/_images/sphx_glr_plot_isolation_forest_002.png (visited on 08/07/2023).

[40] [Online]. Available: https://scikit-learn.org/stable/_images/sphx_glr_plot_isolation_forest_003.png (visited on 08/07/2023).

[41] [Online]. Available: https://img-service.csdnimg.cn/img_convert/369da6030d7e9b29a111c886c7.png (visited on 08/06/2023).

[42] J. Cleland-Huang, A. Agrawal, M. Vierhauser, M. Murphy, and M. Prieto, "Extending MAPE-k to support human-machine teaming," May 2022. [Online]. Available: https://doi.org/10.1145%2F3524844.3528054 (visited on 08/13/2023).

[43] [Online]. Available: https://www.dreamachieversacademy.com/wp-content/uploads/2011/06/plan-ahead-picture.jpg (visited on 08/18/2023).

[44] [Online]. Available: https://assets.datacamp.com/production/tracks/10157/badges/original/Data_Manipulation_2x.png (visited on 08/18/2023).

[45] [Online]. Available: https://ctl.s6img.com/society6/img/6r358r9oXMSCMwUPvybtwLrpQus/w_700/prints/~artwork/s6-original-art-uploads/society6/uploads/misc/17d0e29f5da4490dba72da7f70a59bcc/~~/enable-disable-switch-prints.jpg (visited on 08/18/2023).

[46] [Online]. Available: https://datacontroller.io/ (visited on 08/18/2023).

[47] [Online]. Available: https://www.cisco.com/c/dam/assets/swa/img/anchor-info/what-is-iam-banner-628x353.png (visited on 08/18/2023).

[48] [Online]. Available: https://www.technologyiowa.org/wp-content/uploads/2021/05/Member_Event_IncidentResponse_F.jpg (visited on 08/18/2023).

[49] [Online]. Available: https://cdn1.iconfinder.com/data/icons/messages-line/32/block-message-messages-blocked-512.png (visited on 08/18/2023).

[50] [Online]. Available: https://projects.seattletimes.com/2019/boeing-737-max-12-problems/assets/aoa-sensor-explainer-3.png (visited on 08/18/2023).

[51] S. Borgwardt and P. Koopmann, "Logic-based ontology engineering - ontology maintenance - hitting set tree algorithm," May 2022. [Online]. Available: https://tu-dresden.de/ing/informatik/thi/lat/ressourcen/dateien/lboe18/Part4_1.pdf?lang=en (visited on 08/22/2023).

[52] [Online]. Available: https://www.uavnavigation.com/support/sites/default/files/5-55_edited2.jpg (visited on 08/23/2023).

[53] N. I. of Standards and Technology, "The nist cybersecurity framework 2.0," Aug. 8, 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.ipd.pdf (visited on 08/24/2023).