Dipl.-Ing. Dipl.-Ing. Christopher Preschern, BSc, BSc

Pattern-Based Development of Embedded Systems for Safety and Security

Dissertation

vorgelegt an der
Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften
(Dr. techn.)

durchgeführt am Institut für Technische Informatik
Technische Universität Graz
Gutachter: Em. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Graz, im Juni 2014

Kurzfassung

Verbindungen mit dem lokalen Netzwerk, oder sogar mit dem Internet, von eingebetteten Geräten nehmen ständig zu. Gründe dafür sind zum Beispiel erweiterte Funktionalitäten oder Strategien eines Geräts, wenn es zusätzliche Information von anderen Geräten erhalten kann, oder aber bequemere Bedienung oder Wartung über einen Remote-Wartungstunnel. Von einem Security Standpunkt aus betrachtet, stellen diese Verbindungen der Geräte einen möglichen Angriffspunkt dar. Wenn ein Angreifer es schafft zum Beispiel einen Remote-Wartungstunnel zu übernehmen, dann kann er die Funktionalität des Geräts beeinträchtigen. Das ist insbesondere gefährlich, wenn das Gerät eine personensicherheitsrelevante Funktion erfüllt. Dies sind Funktionen, welche eine Gefahr für die Gesundheit oder das Leben eines oder mehrerer Menschen darstellen. Wenn ein Angreifer ein solches Gerät attackiert, dann ist solch ein Angriff personensicherheitsrelevant. Dennoch beschäftigen sich Zertifizierungsorganisationen und Standards für personensicherheitsrelevante Geräte nicht mit bösartigen Angriffen und Attacken. Diese Standards bieten lediglich Richtlinien zur Personensicherheit an. Sie stellen allerdings keine detaillierten Herangehensweisen um diese Richtlinien zu erreichen, insbesondere nicht unter Berücksichtigung von bösartigen Angriffen und Attacken, zur Verfügung.

Diese Arbeit präsentiert eine Herangehensweise zur Entwicklung von personensicherheitskritischen Geräten unter Berücksichtigung von Informationssicherheit (bösartige Attacken und Angriffe). Die Herangehensweise basiert auf der Verwendung von Design Patterns speziell entwickelt für personensicherheitskritische Systeme. Diese Design Patterns wurden aus der Literatur gesammelt und aus dem IEC 61508 Personensicherheitsstandard erhalten. Um Nachweise über entwickelte Personensicherheitsmaßnahmen zu geben, sind die Design Patterns mit Diagrammen ausgestattet, welche die Sicherheitsmaßnahmen die bei Anwendung des Patterns eingesetzt werden strukturiert präsentieren. Für diese Diagramme wurde Goal Structuring Notation (GSN) verwendet. Die GSN Diagramme der Patterns setzen das Personensicherheitsziel des Systems in Verbindung mit konkreten Maßnahmen zu dessen Implementierung. Diese Maßnahmen beruhen auf Methoden, welche explizit im IEC 61508 Standard vorgeschlagen werden. Die Patterns beinhalten also eine direkt auf dem IEC 61508 Standard beruhende Argumentation für die Personensicherheit. Aus Sicht der Informationssicherheit sind die Patterns ebenfalls mit einem GSN Diagramm ausgestattet, welches zeigt warum das System gegen Angriffe sicher ist. Dazu wurden für die Patterns mögliche Bedrohungen mit der STRIDE Security Analyse festgestellt. Bedrohungen, welche die personensicherheitskritische Funktionalität beeinflussen können, werden in dem GSN Diagramm dargestellt. Die Design Patterns beinhalten also Herangehensweisen für die Implementierung von personensicherheitskritischen Systemen unter Berücksichtigung von Informationssicherheit. Für Personen- und Informationssicherheit beinhalten die Design Patterns GSN Diagramme um Argumentationen zu liefern die diese Sicherheiten belegen.

Abstract

Embedded devices are becoming more and more interconnected. Such interconnections (e.g. via the Internet) bring advantages in terms of more manageable control and maintenance or in terms of more effective strategies of the devices possible due to additional information. However, from a security point of view, the interconnections also considerably increase the attack surface of these devices. For example, if an attacker can manipulate the maintenance channel of an embedded device, the attacker can usually take control of the device and affect its functionality. This is especially dangerous if the functionality is critical. For example, safety-critical systems are systems whose malfunction poses a threat to human health of even human lives. If an attacker tampers with such a system, the attack can pose a threat to safety and thus such attacks should be considered as safety-critical. However, current safety certification organizations and safety standards do not cover security concerns. The standards usually state requirements simply from a safety point of view and do not give guidance on how to achieve these requirements, especially when related to security.

This thesis provides such guidelines in the form of safety design patterns which also cover security concerns. The safety patterns are gathered from literature or mined from the IEC 61508 safety standard. To enable safety reasoning, the patterns are equipped with a Goal Structuring Notation (GSN) diagram to connect the overall goal of applying the pattern to the actual implemented methods. These methods are based on the IEC 61508 standard. Thus, the design patterns build their overall GSN safety argument on methods suggested for use by the IEC 61508 safety standard. From a security point of view, the pattern threats are gathered with a STRIDE security analysis and the resulting threats are organized in a GSN diagram containing all threats which could affect the safety functionality. Thus, the patterns contain safety-related solutions which consider security aspects. The patterns provide GSN diagrams containing arguments for the overall system's safety and security.

Acknowledgements

I would like to thank by girlfriend Silke and my family for tolerating me and for keeping me going during my studies. They helped to motivate me and cheered me up during tough and stressful periods of my PhD study. Thank you very much!

This thesis has been conducted as part of the Andritz Hydro HIPASE project. First of all, I would like to thank Christian Kreiner for managing the cooperation with Andritz Hydro. He initiated the collaboration and convinced me to do my PhD as part of the HIPASE project. Christian Kreiner was my mentor and helped me with technical and organizational issues encountered with my thesis. He also introduced me to the topic of design patterns and suggested that I study such patterns for the application in the field of safety in my dissertation. This helped guide me towards my actual PhD topic and enormously decreased the "fuzziness" of my work.

I would like to thank the supervisor of my thesis, Prof. Reinhold Weiß. He made the official and organizational part of my PhD study very easy by being open regarding the topic of my thesis and by giving me a lot of freedom regarding my research topics and the research communities I joined. Prof. Weiß also spent a lot of time helping me refine the topic of my thesis and helping me to elaborate it from a scientific point of view.

From the Institute for Technical Informatics, I would also like to thank all my colleagues and all the staff who made my work days easier and more enjoyable. In particular, I want to thank my colleagues also working on the HIPASE project: Nermin Kajtazovic and Andrea Höller. They both kept the HIPASE project running by putting a lot of effort into keeping Andritz Hydro satisfied by providing them concepts and implementations. This allowed my to not be swamped with projects related work and thus allowed me to focus on the research part of my thesis. The same holds for all my students who worked on projects related to the HIPASE project.

I would like to thank Andritz Hydro for making my PhD study possible by cooperating with the Institute for Technical Informatics and providing me with paid employment during my PhD study. I also want to thank Andritz Hydro for letting me participate in the HIPASE project by bringing in new ideas. It was essential for my PhD thesis to get feedback from Andritz Hydro on how safety development works in practice and on how the ideas I brought in worked out. In particular, I want to thank Rudolf Neuner from Andritz Hydro. He was my main contact person at the company and was very open to my ideas. Furthermore, he allowed me to spend enough time on research related aspects of my work which made it a lot easier for me to complete my PhD study quite quickly.

During my research activities I got in touch with the pattern community and I would like to thank all of them for providing me with lots of valuable feedback on my work and for making the days at the pattern conferences an enjoyable and memorable time.

Graz, June 2014 Christopher Preschern

Extended Abstract

The increasing connectivity of embedded systems leads to new opportunities as well as challenges. If such systems are connected via the Internet or via a local network, they can interact with each other and can perhaps better achieve their purpose by taking a more holistic approach through utilizing information or services from connected devices. However, a major drawback of interconnected embedded systems is that such connections, especially when realized via the Internet, open the door for attackers to compromise the functionality of the embedded system. Such attacks on embedded systems are particularly problematic if the system has to fulfill critical functionality. For example, if a malfunction of the system poses a threat to human health or even human life, such a system is considered as safety-critical and it has to be assured that it is very unlikely that such a system will malfunction. Malfunctions can be caused by malicious attacks as well as accidental failures of the system. Such accidental failures can for example be caused by hardware faults or by faults made during system development. To ensure that accidental faults are rather unlikely, safety-critical systems are usually certified according to safety standards. For general embedded systems, the IEC 61508 safety standard is commonly used in Europe. This standard defines a safety development lifecycle and consists of requirements for different activities within this lifecycle as well as requirements for software development approaches and for the hardware which is used. However, the standard does not provide much guidance on how to achieve the given requirements or how to provide evidence that the requirements have been achieved. Furthermore, the IEC 61508 safety standard does not address security issues, so malicious attacks as a cause for safety-critical malfunctioning of the system are not considered. However, particularly with increasingly connected systems, this source of malfunction poses an increasing threat for the integrity of safety-critical systems. Also especially when safety- and security-critical aspects are combined, there is little guidance in the safety standards or other literature on how to develop such systems.

For guidance on how to design and develop safety- or security-critical systems, patterns can serve to document and provide expert knowledge. Patterns were first introduced by Christopher Alexander in his book *"A Timeless Way of Building"* [Ale79]. Alexander's patterns provide commonly applied quality solutions to relevant problems related to architecture and construction of buildings and cities. The idea of patterns was adopted to the software domain, where patterns for constructing and designing good software were described. Applying these patterns to the development of software became best practice and from that point on, patterns became more popular and also known to other domains. For example, patterns exist for safety-critical as well as for security-critical systems. However, the interaction between safety and security is still an open topic and has not yet been extensively covered in pattern or other literature.

To address this matter, this thesis collects safety patterns. The patterns are gathered from literature and identified from the IEC 61508 safety standard. The patterns are presented in a uniform notation containing a diagram to describe the connections and data flows of the software and hardware elements. Based on this diagram, a security analysis is conducted which results in all relevant threats for the patterns. In particular, the focus is on the threats which are relevant for an attacker wishing to compromise the safety functionality of the system. Thus, the collection of patterns addresses safety problems whilst keeping security aspects in mind.

It is relevant for both safety and security to gather evidence on whether a system design decision, such as the application of a pattern, achieves or helps the overall safety or security goal. Such evidence has to be provided for safety if the system has to be safety certified. During safety certification, an assessor has to be able to follow the presented evidence and see how the safety goals are addressed by the design and implementation.

Such evidence arguments can be included in the patterns. To provide these evidence arguments, Goal Structuring Notation (GSN) is used. GSN is a standardized notation used in the safety domain to structurally connect goals to evidence data confirming or indicating the goal's achievement. For the safety patterns, GSN connects an overall safety goal which should be achieved when applying a pattern to the actual design and implementation approach that is taken to address the problem. The safety patterns contain GSN skeletons containing arguments as to why the system is safe when applying the patterns as well as arguments as to why the system is secure.

- For the safety argument, the patterns provide links to specific safety approaches explicitly suggested for use by the IEC 61508 standard. These approaches provide developers with detailed information on how to implement the pattern. On the other hand the patterns contain a safety argument based on approaches provided by the safety standard.

- For the security argument, we present the analyzed security threats which could affect system safety in a security GSN diagram.

Thus, this thesis provides safety patterns which present their strength and weaknesses in terms of safety and security arguments which can be used as evidence for system certification. When applying the patterns, a developer has to choose an appropriate safety pattern. Based on this pattern, the developer gets skeletons for the safety and security GSN argument. For the specific architecture where the pattern is applied, the developer has to complete the safety and security GSN diagrams to obtain a complete safety and security representation of the system. The process of this pattern application is shown in Figure 1. The three main contributions of this thesis are the following:

1. The thesis presents a collection of safety patterns from literature in a uniform notation and provides the connections between the patterns to structurally build a safety system. Compared to a simple pattern collection, our pattern system brings the benefits of providing an overview of the safety patterns and their relationships. This makes the selection of patterns easier.

2. The thesis analyzes security aspects of the safety patterns. The inclusion of security threats based on a structured security analysis in the safety patterns enables one to

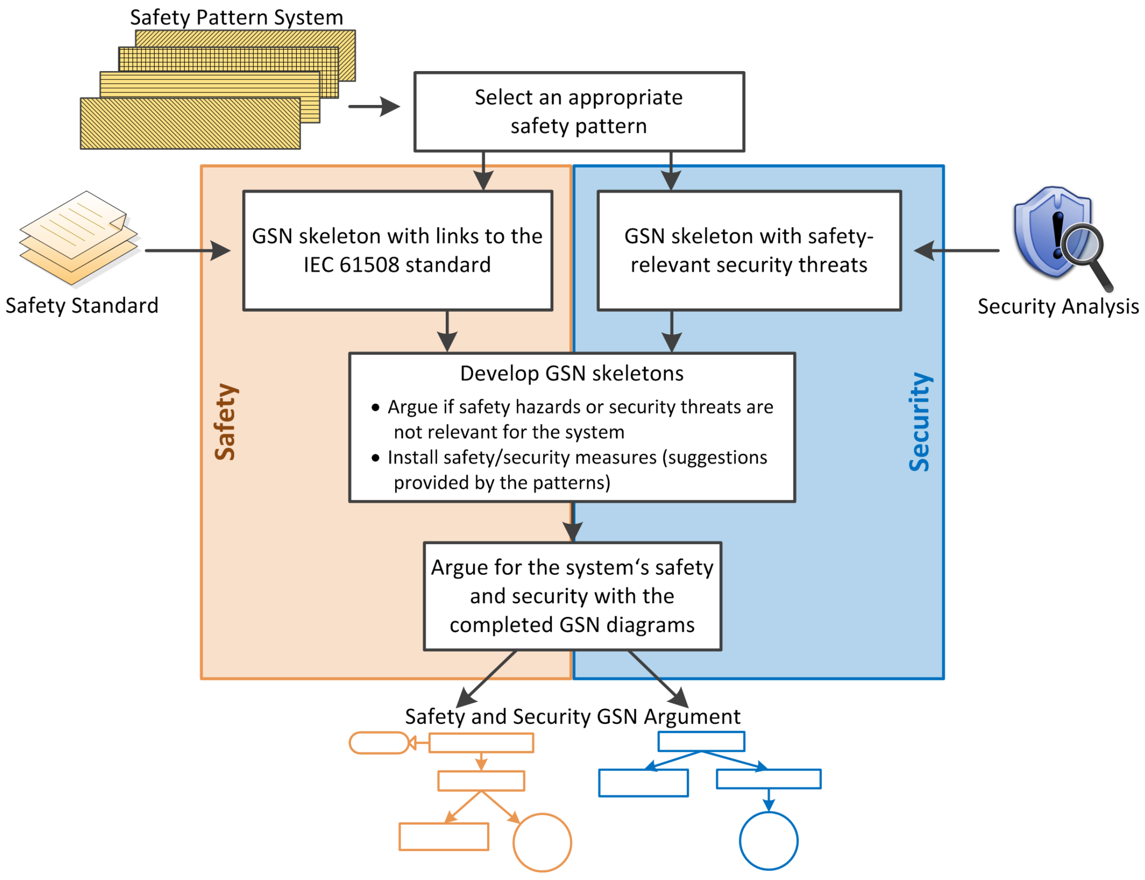

Safety Pattern System
Select an appropriate safety pattern
Safety Standard
GSN skeleton with links to the IEC 61508 standard
GSN skeleton with safety-relevant security threats
Security Analysis
Safety
Security
Develop GSN skeletons
• Argue if safety hazards or security threats are not relevant for the system
• Install safety/security measures (suggestions provided by the patterns)
Argue for the system's safety and security with the completed GSN diagrams
Safety and Security GSN Argument

Figure 1: Design process for applying the safety patterns. After selecting an appropriate pattern, the safety and security arguments have to be further developed for the specific architectures to obtain a complete safety and security argument

consider security at an early stage in the system design. Additionally, the security threats increase the awareness of security problems in the safety domain even if they are not yet required by the safety standards.

3. The thesis equips the safety patterns with safety and security arguments in the form of GSN diagrams. These diagrams link the design decision from a safety point of view to the IEC 61508 standard and, from a security point of view, they provide a method for presenting safety-relevant security threats and they provide information on how to mitigate these security threats.

Including security consideration during early system development by using the safety patterns with their security GSN arguments brings the advantage of being able to build more robust systems by realizing security threats early on and by building in security measures from start. The patterns considering safety and security also allow a system designer to evaluate trade-offs or synergies between safety and security when making safety-related architecture decisions.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.1.1 Emerging Security Requirements for Safety-Critical Systems . . . 1
1.1.2 Lack of Security Considerations in Safety Standards . . . 3
1.2 Applying Safety Patterns for the Development of Critical Systems . . . 3
1.2.1 The HIPASE Project . . . 4
1.2.2 Problem Statement . . . 5
1.2.3 Contributions and Significance . . . 6
1.2.4 Structure of the Work . . . 7

2 Related Work **9**
2.1 Security in Safety Standards . . . 9
2.1.1 Overview of Safety Standards . . . 9
2.1.2 Security in IEC 61508 and Related Standards . . . 10
2.1.3 Security in other Safety Standards . . . 13
2.1.4 Standards and Reports on Integrated Safety and Security . . . 14
2.1.5 Discussion . . . 18
2.2 Safety Patterns . . . 18
2.2.1 Introduction to Patterns and Tactics . . . 18
2.2.2 Safety Patterns from Literature . . . 19
2.2.3 Pattern-Based Safety Development . . . 19
2.2.4 Discussion . . . 22
2.3 Security Analysis of Patterns . . . 22
2.3.1 Security Analysis of Security Patterns . . . 22
2.3.2 Application of Security Analyzed Patterns . . . 23
2.3.3 Discussion . . . 24
2.4 Structuring Evidence for Safety Compliance . . . 24
2.4.1 Safety Case . . . 24
2.4.2 Boolean logic Driven Markov Processes . . . 27
2.4.3 Fault Trees . . . 27
2.4.4 Discussion . . . 28
2.5 Summary and Difference to the State-of-the-Art . . . 28

3 Safety Patterns with Security Aspects **31**
3.1 Selection of Safety Patterns . . . 32
3.2 Selection of a Pattern Form . . . 33
3.3 Safety GSN . . . 33
3.3.1 Scenario Mining . . . 34
3.3.2 Analyzing Safety Tactics for the Patterns . . . 34
3.3.3 Building the Safety GSN based on the Tactics and Scenarios . . . 35

3.4 Security Aspects of Safety Patterns . . . 36
3.4.1 Threat Analysis . . . 36
3.4.2 Building the Security GSN based on the Threats . . . 39
3.5 Organizing the Safety Patterns to a Pattern System . . . 40

4 Application of the Patterns to a Case Study 43
4.1 Case Study Description . . . 44
4.2 Pattern Selection . . . 45
4.3 Application of the Pattern . . . 46
4.4 Construction of a Safety Argument . . . 47
4.4.1 GSN from the Pattern . . . 47
4.4.2 Choosing appropriate IEC 61508 Methods related to the Tactics . . . 48
4.4.3 Elaborating the Safety GSN Diagram . . . 48
4.5 Construction of a Security GSN Diagram . . . 50
4.5.1 GSN from the Pattern . . . 50
4.5.2 Elaborating the Security GSN Diagram . . . 50
4.5.3 Influence on the Safety Functionality . . . 52

5 Evaluation of the Patterns and the Pattern-Based Development Method 55
5.1 Discussion and Benefits for the HIPASE Project . . . 55
5.2 Comparison to other Pattern-Based Safety Development Methods . . . 57
5.3 Evaluation of the Safety Patterns . . . 60
5.3.1 The Safety GSN Diagrams . . . 60
5.3.2 Link to the IEC 61508 Standard . . . 60
5.4 Evaluation of the Security Aspects . . . 61

6 Conclusion and Future Work 63
6.1 Conclusion . . . 63
6.2 Directions for Future Work . . . 64
6.2.1 Extend Pattern Catalog . . . 64
6.2.2 Towards a Pattern Language . . . 65
6.2.3 Security Certification . . . 65
6.2.4 Evaluation . . . 65
6.2.5 Tool Support . . . 66

7 Publications 67
7.1 Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle . . . 69
7.2 Building a Safety Architecture Pattern System . . . 87
7.3 Security Analysis of Safety Patterns . . . 115
7.4 Safety Architecture Pattern System with Security Aspects . . . 127
7.5 Pattern-Based Safety Development Methods: Overview and Comparison . . . 175
7.6 Applying and Evaluating Architectural IEC 61508 Safety Patterns . . . 195
7.7 Catalog of Security Tactics linked to Common Criteria Requirements . . . 201
7.8 Quantitative Security Estimation based on Safety Architecture Design Patterns . . 219
7.9 Structuring Modular Safety Software Certification by Using Common Criteria Concepts . . . 225

References 229

List of Figures

1 Design process for applying the safety patterns. After selecting an appropriate pattern, the safety and security arguments have to be further developed for the specific architectures to obtain a complete safety and security argument vi

1.1 Number of reported security incidents on automation systems taken from [MR12]. The increasing number suggests that security is becoming an increasingly important topic. 2
1.2 The four core functionalities of the hydro-electric power plant devices: Turbine control, Protection, Synchronization, and Excitation (old hardware generations before HIPASE shown in the picture) 4
1.3 Recent attacks on different industrial automation sectors. Most of the attacks are targeted at devices in the energy sector. [NWD$^+$12] 5

2.1 Selected Safety Standards based on the IEC 61508 Standard [SS04]. 9
2.2 The SeSa Method. This method is used to analyze the security of remote-accessible safety-systems. [GJØO10] 15
2.3 The SafSec Methodology. The main aims are to achieve combined safety and security certification as well as modular certification. [LCJ05] 16
2.4 The SQUALE Evalution Framework. The boxes describe activities during this process and the arrows represent quality assuring activities. [Cor99] 17
2.5 Claim Argument Evidence Notation. The figure shows how a generic claim is fulfilled by evidence elements which are connected to the claim via arguments. [BB98] . . . 25
2.6 Claim Argument Evidence Notation. The figure shows the main elements and their notation for building a CAE diagram. 25
2.7 Goal Structuring Notation. The figure shows the main elements and their notation for building a GSN diagram. 26
2.8 Boolean logic Driven Markov Processes. Example model for the security of a Remote Access Server system. The dashed lines represent the triggers. [PCB10a] 28

3.1 Overview of steps described in this chapter to build a safety pattern system with security aspects from safety patterns in literature (based on **Publication 4**). . . . 31
3.2 Graphical representation of the components from the Homogenous Duplex Pattern. If a primary channel fault is detected by the fault detector, the switch switches to the output of the backup channel (based on **Publication 2**). 32
3.3 Architectural tactics for safety. The tactics are grouped into principles which avoid failures, which detect failures, and which contain failures (based on **Publication 1**). 34
3.4 Tactic representation of the Homogenous Duplex Pattern (based on **Publication 2**). 35
3.5 Safety GSN skeleton of the Homogenous Duplex Pattern (based on **Publication 4**). 36

3.6 Security GSN skeleton of the HOMOGENOUS DUPLEX PATTERN (based on **Publication 3**). Not all, just safety-relevant threats are included. They are represented as undeveloped goals and have to be elaborated by a devleoper. 39
3.7 Overview of all patterns of the safety pattern system. The structurally identified relations are shown by arrows between the patterns. 41

4.1 Overview of steps described in this chapter to apply a safety pattern system with security aspects (based on **Publication 4**). 43
4.2 Basic system architecture and interfaces of the HIPASE-device not yet considering safety 45
4.3 Main elements of the HOMOGENOUS DUPLEX PATTERN. Instead of a switch, for this pattern, a voting element can also be used for whichever requires two correct inputs, or one correct input and one from the fault detector excluded channel. . . . 46
4.4 HIPASE-device architecture after applying the HOMOGENOUS DUPLEX PATTERN. The switch component of the pattern has been replaced by the hardware voting functionality realized as relays. 47
4.5 Safety GSN diagram of the HOMOGENOUS DUPLEX PATTERN. The GSN contains the tactics applied by the pattern: *Replication Redundancy*, *Override*, and *Condition Monitoring* 48
4.6 Elaborated Safety GSN diagram for the HIPASE-device architecture. All the tactics are replaced by actual IEC 61508 methods and all goals are connected to evidence elements. 50
4.7 Security GSN skeleton of the HOMOGENOUS DUPLEX PATTERN. The GSN diagram contains security threats which could influence the safety functionality. 51
4.8 Completed Security GSN argument for the HIPASE-device. No more GSN goals are undeveloped and all goals are linked to evidence elements. 52

7.1 Overview of the process describing the development, application, and evaluation of the safety patterns with security aspects. The figure includes relevant publications of the author of this thesis and shows which part of the process the publications address. 68

List of Tables

2.1 Overview of literature which introduces safety-related or fault tolerance patterns (Table based on **Publication 2**) 20

3.1 STRIDE threats mapped to the four different types of data flow diagram elements. 37
3.2 Considered STRIDE threats for the entities of the graphical diagrams which are part of the safety pattern descriptions. 38
3.3 Description of pattern relations (slightly modified from [KP10b]) 40

4.1 Tactics and their related IEC 61508 methods. The column on the right shows a checkmark for all methods that are selected to be applied by the HIPASE-device architecture (Table partially based on **Publication 1**) 49

5.1 Comparison of pattern-based safety methods with focus on their safety and security aspects (based on **Publication 5**) 58
5.2 Precision and recall values for the IEC 61508 method suggestion of the Homogenous Duplex Pattern 61
5.3 Security risk improvement values for the HIPASE-device architecture when applying three initially considered safety patterns 62

Chapter 1

Introduction

1.1 Motivation

For embedded devices which interact with the physical world and could perhaps even cause physical damage, it is not just important that they simply perform their functionality. They have to perform their functionality every time, under all circumstances, and in any imaginable condition. Developing such devices causes several challenges. These include providing evidence that the device actually performs as intended, particularly if security concerns have to be considered. This is especially relevant for modern interconnected embedded devices.

1.1.1 Emerging Security Requirements for Safety-Critical Systems

As of today, several embedded systems are interconnected via local networks or via the Internet. These connections enable more effective strategies due to additional information from the Internet or from other embedded devices. Another advantage of this connectivity is that the operation and maintenance of embedded systems can be made more straightforward. As an example, for industrial control systems it is common practice to have a remote connection to the control devices for maintenance purposes. Usually such a remote connection is realized with a VPN tunnel. Such a VPN tunnel is, from a theoretical security point of view, currently secure; however, practically there are security issues because of possible faults in the software implementation of the VPN tunnel or because of human factors such as leaked passwords. When an attacker manages to take control of such a maintenance tunnel for an embedded device, it is not usually a problem for the attacker to manipulate the functionality of that device, because commonly the devices were not built with defense in depth in mind. Thus, when breaking into the system, an attacker can usually take full control over the device. Figure 1.1 shows that security threats such as this are an emerging problem. The figure presents the number of reported major security incidents experienced by industrial automation systems over recent years. Even though the overall number of incidents is quite low, one can see a trend that the number of incidents is increasing. The low number of overall incidents can be explained due to the fact that these are just the reported incidents. The number of actual incidents is assumed to be a lot higher.

The recent example of the *Stuxnet* attack [Lan11] showed that the security threat

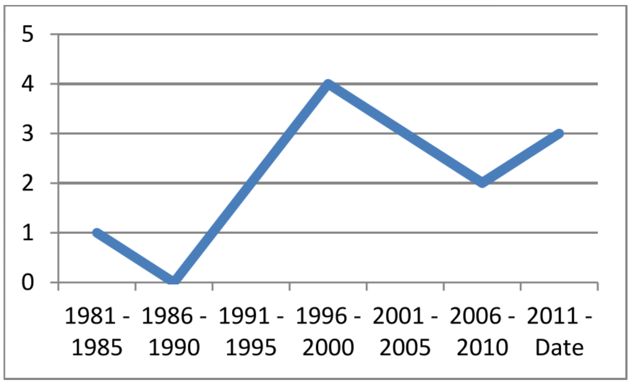


Figure 1.1: Number of reported security incidents on automation systems taken from [MR12]. The increasing number suggests that security is becoming an increasingly important topic.

for embedded systems, even if they are not directly connected to the Internet, is real. The Stuxnet attack was targeted at Iranian uranium enrichment plants operating with Siemens industrial control devices. The attack was very sophisticated and introduced malicious software into the control software even though the control devices were not directly connected to the Internet. Not directly connecting embedded devices to the Internet or to the local network has so far been common practice for critical systems, because this measure is supposed to block out any attacks. While not connecting the device to the Internet is a very good measure, apparently it is not sufficient to be resistant to today's security threats. In particular for systems with safety-critical functionality, where a malfunctioning system poses a threat to human health or even to human lives, it is important to be resistant to commonly known security attacks and to be as far as possible resistant against any other anticipated attacks. This is an important topic for safety-critical systems, because if an attacker gets access to the system, the attacker can also most likely influence the safety-related functionality of the system. Therefore, some security issues have to be considered as safety-critical. Up until now most safety-critical devices are not connected to public networks where they are exposed to attacks; however, that trend is changing and thus the attack surface for safety-critical system drastically increases.

As seen in the Stuxnet incident, simply not connecting the device to the public Internet can be seen as a very good measure to greatly decrease the possibility of attacks, but it cannot completely prevent attacks. To prevent more sophisticated attacks, a system should have further security measures and provide defense-in-depth. For the above mentioned example of the maintenance tunnel, a system could in addition to the security measures provided by the VPN tunnel be equipped with, for example, intrusion detection measures that would enable it to be resistant against attacks to some extent even if a security measure has been breached [PKK12]. However, several of such additional security measures require design from the start and cannot be post-engineered. Therefore, it is important to consider security from the start of the design process and during the whole development process of safety-critical systems.

1.1.2 Lack of Security Considerations in Safety Standards

For safety-critical systems there are, depending on the domain of the actual system, several different standards. It is industrial practice to certify a safety-critical product according to the relevant standard on the one hand to comply with laws and on the other hand to show that the product meets certain quality and in particular safety criteria. For general embedded systems, the IEC 61508 safety standard is commonly used in Europe. For more specific domains (such as automotive or railway), there are more specific safety standards available of which some are based on the IEC 61508. The IEC 61508 safety standard contains requirements for the system's development process and requirements for the hardware and for software development. For example, for software it has to be ensured that the probability of software failures arising due to development faults is sufficiently low. However, the possibility of software faults arising due to malicious attacks is not considered by the standard.

This blind spot of current safety standards is considered as a great weakness of safety systems [Tem11]. There are already proposals for standards which consider both, safety and security. However, these are just proposals and in industrial practice they are not applied. This means that for current safety-critical systems, implementing security measures is not mandatory to achieve safety certification although security attacks could tamper with safety functionality.

Alternatively, security certification for such safety-critical systems could provide a countermeasure for cyber-attacks. However, again it is not currently common practice to undergo security certifications in safety industries. Thus, usually when a safety-critical product is developed, it is just certified according to safety standards and there is no regulatory authority which checks the product's compliance with security demands. This leaves it open to the developer of the safety-critical system to decide to which extent security issues will be considered and quite often developers from the safety domain are not familiar with security issues and methods.

1.2 Applying Safety Patterns for the Development of Critical Systems

To address the challenges regarding design guidelines for safety and security critical systems, this thesis proposes a pattern-based development approach which results in safety arguments related to the IEC 61508 safety standard as well as security arguments. The safety and security arguments are constructed with Goal Structuring Notation (GSN) diagrams. GSN is a structured notation used in the safety domain to build and represent arguments showing how general goals (e.g. *"The system is safe and secure"*) are achieved through different evidence elements. The integration of GSN diagrams into safety patterns, the analysis of security aspects of these patterns, and the approach of applying the patterns will be described in this thesis in general and with the application to an industrial case study: The HIPASE project.

1.2.1 The HIPASE Project

This thesis is conducted as part of the HIPASE project. HIPASE is a product family of hydro-power plant controllers developed and produced by Andritz Hydro. The HIPASE devices will control mechanical components (such as turbines) of hydro-electric power plants. The four main elements of HIPASE devices are shown in Figure 1.2 and they are:

- *Turbine control* - has to control the speed of the turbines through magnet valves
- *Protection* - monitors voltages, currents, and mechanical components in the hydro-power plant and makes sure that devices are switched off before they incur any damage
- *Synchronization* - controls and makes sure that the voltage, frequency, and phase is synchronous to the power grid before connecting to it
- *Excitation* - controls the current flowing through the generator and monitors the temperature

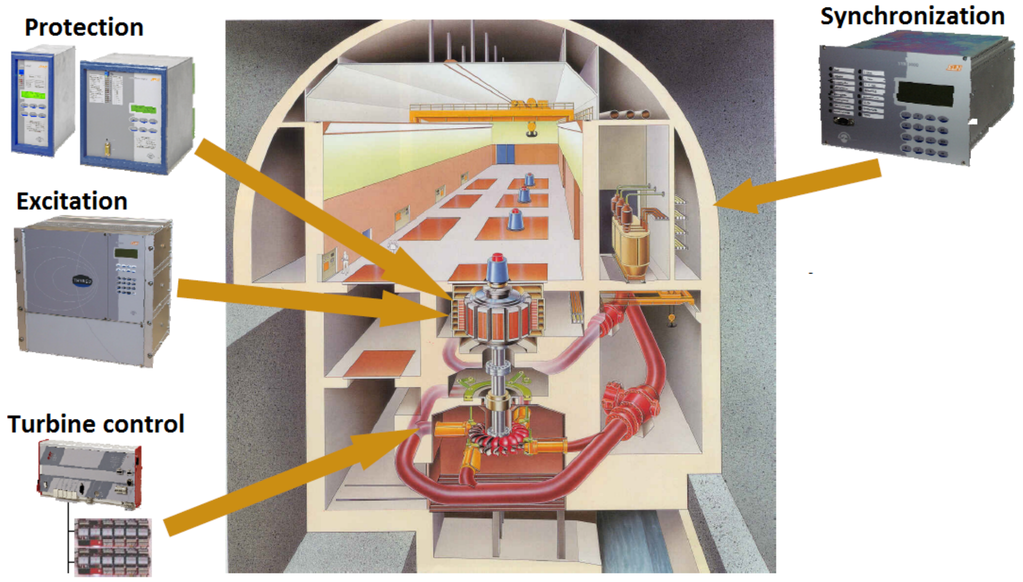


Figure 1.2: The four core functionalities of the hydro-electric power plant devices: Turbine control, Protection, Synchronization, and Excitation (old hardware generations before HIPASE shown in the picture)

Up until now, four different products of Andritz Hydro were responsible for these four functionalities. Now, the HIPASE device should cover all four functionalities. The HIPASE devices will use the same hardware platform, but can be programmed to operate a different function. Thus, in a hydro-electric power plant there might be more than one HIPASE device; however, they are programmed differently to operate on different functions such as protection or excitation.

The HIPASE device is safety-critical, because malfunctions could lead to physical damage to the equipment (e.g. the turbine) and to people near that equipment. Therefore, the HIPASE product will be SIL3 certified according to the IEC 61508 safety standard. In addition to this safety certification, Andritz Hydro wants to put a strong focus on security, because the HIPASE devices interact with each other via the local hydro-power plant network more intensively than the older hardware generation of Andritz Hydro did. Therefore, for the HIPASE device, Andritz Hydro wants to build security in from the start of the product design. This decision can be seen as a very positive one, because recent research shows that the energy sector is a very critical one and subject to more attacks than other industrial sectors (see Figure 1.3). This indicates that introducing security measure into hydro-power plants is very reasonable, however, still visionary for the energy sector.

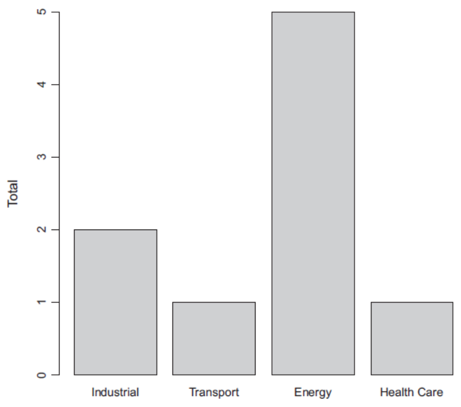


Figure 1.3: Recent attacks on different industrial automation sectors. Most of the attacks are targeted at devices in the energy sector. [NWD+12]

A very interesting point for the HIPASE project is to suggest appropriate safety architectures, to consider their impact on security, and, if appropriate, to suggest security measures to protect against possible attacks and to provide defense-in-depth. However, if possible, these security measures should not interfere too much with safety-critical components of the HIPASE system, because if the core-safety functionality can be separated from additional security features, these features do not have to undergo safety certification which is highly desirable due to cost reasons.

1.2.2 Problem Statement

The connection of safety-critical systems to other embedded devices or even to the Internet exposes these systems to attacks and requires special attention with regards to security aspects. Furthermore, the interplay between introduced security aspects and safety functionality as well as safety certification yields challenges regarding the ability to provide structured evidence regarding safety- and security-assurance which can be presented for system certification. The following provides a more detailed overview of the challenges mentioned:

- Safety standards, in particular the IEC 61508 standard used in the HIPASE project, state many requirements regarding product design and development. Additionally, the IEC 61508 standard provides a pool of specific methods (e.g. *Software diversity* *(diverse programming), Cross-monitoring of multiple actuators*) which are recommended for use, depending on the product's level of safety criticality. However, the IEC 61508 standard does not provide guidance on which of these methods should actually be used to achieve specific safety goals. Therefore, someone new to the safety standard is confronted with a huge set of requirements and a huge set of unrelated methods. This results in poor guidance on how to select appropriate methods.

- For the selection of methods from the IEC 61508 standard, it should be traceable why they were selected and which safety goal they influence. The IEC 61508 standard does not provide approaches for such structured safety reasoning. However, if a system has to be safety certified, one has to provide safety evidence to the assessor. Thus, an appropriate approach for presenting safety arguments for the system which can be produced for certification is highly desirable.

- Safety standards do not currently cope with security aspects. The most recent version of the IEC 61508 standard mentions that security can be an issue and provides vague recommendations such as: *"If security threats have been identified, then a vulnerability analysis should be undertaken in order to specify security requirements"* [Int10]. However, the standard does not provide specific goals, requirements, or methods from a security point of view, even though security might also be an issue for the safety functionality of the system. Still, for the development of a modern safety-related system, security is highly relevant. However, as security aspects are left out by safety standards and safety-related products usually do not have to be security certified, security aspects are rather often not appropriately attended to. At this point, integrating security aspects into existing or well-known safety measures to increase the awareness of security problems for safety-critical systems could help to bring security-thinking into the safety world.

In this thesis, the above mentioned shortcomings regarding the safety standard and the common practice safety system development have been addressed through security enhanced safety patterns. A set of safety patterns is collected, organized, and structurally presented. The patterns are extended to contain links to applicable IEC 61508 methods and they contain Goal Structuring Notation (GSN) diagrams to structurally present how the selected IEC 61508 methods achieve a system's safety goal. From a security point of view, the patterns are analyzed regarding their security threats and those threats which are relevant for the system's safety are represented in a GSN diagram. Thus, in addition to the benefit of reusing well-proven solutions through the application of patterns, a safety engineer gets safety and security reasoning methods in the form of GSN diagrams included in the patterns.

1.2.3 Contributions and Significance

In summary, the thesis provides contributions in the following fields:

1. **Safety Pattern System**
   The thesis collects architectural safety patterns based on pattern literature and based on the IEC 61508 standard. The patterns are represented in a uniform notation and

their relationships are presented to provide a good overview on the patterns for pattern selection.

2. **Pattern Connection to the Safety Standard.**
   The safety patterns are connected to IEC 61508 methods. Each pattern contains related architectural design decisions (safety tactics) contained in the pattern and the tactics are linked with specific IEC 61508 methods. Thus, a developer gets specific information on how to implement the pattern from the safety standard.

3. **Safety Argument**
   The safety patterns are equipped with GSN diagrams containing selected IEC 61508 methods applied by the patterns. The GSN diagrams contain information on how the methods are used to meet the overall safety goal that should be achieved by applying a pattern. Thus, the patterns provide the basis for structured safety reasoning.

4. **Security Threats**
   The safety patterns are all presented in a uniform notation containing a diagram showing the pattern's hardware and software components and their data flows. Based on these diagrams, a structural security analysis is conducted to provide security threats for the patterns.

5. **Security Argument**
   Based on the pattern's security threats, the ones which could influence the safety functionality of the system are selected and presented in a GSN diagram. This diagram provides safety engineers with an overview of highly critical threats for the system and provides a basis for a structured argument on why security threats are sufficiently handled in order to protect the system's safety functionality.

1.2.4 Structure of the Work

The thesis is structured as follows:

Chapter 2 describes related work. The related work covered concerns design patterns in general and design patterns specific for safety and security as well as pattern-based development. Furthermore, related work on safety and security reasoning and related work on safety and security standards is covered.

Chapter 3 explains how the safety patterns are gathered and organized. This chapter also explains how the safety patterns are related to the IEC 61508 standard and how security aspects are introduced into the patterns. In addition, the chapter describes how security and safety aspects are collected in GSN diagrams. The general approach of how to apply the patterns and how to use the GSN diagrams to build arguments for safety and security is also described.

In Chapter 4, the patterns are applied to a case study (the HIPASE project). the chapter explains the basic HIPASE architecture, and shows how the application of a safety pattern helps to build a resulting architecture including arguments for safety and security.

The patterns themselves as well as the presented pattern-based development approach are evaluated in Chapter 5 with the help of metrics for the patterns as well as with the help of qualitative comparison to existing approaches and with the help of feedback from the HIPASE project.

Chapter 6 concludes this thesis and provides an outlook on possible future work related to the safety patterns and related to their security aspects.

Finally, Chapter 7 presents a selection of publications on which this work is based. The publications give more detailed information about the different steps for building, applying, and evaluating the presented patterns.

Chapter 2

Related Work

2.1 Security in Safety Standards

This section presents relevant safety standards and discusses to which extent they consider security aspects which could affect the functionality of safety-critical systems.

2.1.1 Overview of Safety Standards

When developing systems which pose a threat to human health or even to human lives, such systems are usually safety-certified. Depending on the specific domain, there are several different safety standards which are relevant. For example, in Europe when developing a system it is not mandatory to fulfill any safety standard; however, it is mandatory to fulfill the Machinery Directive 2006/42/EC [Eur06]. Alongside the machinery directive, more specific safety standards were developed which describe more specific requirements. The safety standard targeted at electrical, electronic, and electronic programmable systems in general is the IEC 61508 standard. Based on this standard, other standards for more specific domains (e.g. automotive, nuclear power plants) were developed. Figure 2.1 shows safety standards based on the IEC 61508 which will be addressed in the following sections.

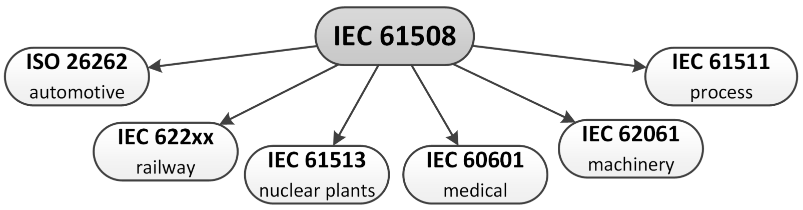


Figure 2.1: Selected Safety Standards based on the IEC 61508 Standard [SS04].

The specific safety standards are process oriented and pose several requirements regarding the development lifecycle and the development phases of a safety-related system. To ensure that the goals of different development activities are met, evidence artifacts have to be produced so that an assessor can understand whether the developed product meets the requirements of the safety standard [PR13].

By producing such relevant artifacts for an assessor to fulfill the domain specific safety standard, one meets the machinery directive. Achieving safety certification according to the domain-specific standard is usually much easier than directly trying to meet the machinery directive requirements. Therefore, in Europe, it is common practice to look for the safety standard most applicable for the specific domain (e.g. ISO 26262 for automotive) and to fulfill the requirements of this standard in order to meet the machinery directive [SS04].

2.1.2 Security in IEC 61508 and Related Standards

As the work in this thesis is related to the IEC 61508 standard, this section investigates in particular security-related aspects of the IEC 61508 and its derived standards.

IEC 61508

The first version of the IEC 61508 standard from 1998 does not mention security aspects of safety systems at all. Version 2 of the standard, which was published in 2010, mentions security aspects, but does not deal with them. The standard explicitly says that it does not cover requirements or policies regarding security which might be needed for safety-related systems. For example, the standard says that a safety manual must include *"any security measures that may have been implemented against listed threats and vulnerabilities"* [Int10], but it does not specify what the measures have to look like and how rigorously they have to be documented in the safety manual.

Other examples where the IEC 61508 standard mentions security are:

- *"If the hazard analysis identifies that malevolent or unauthorised action, constituting a security threat, as being reasonably foreseeable, then a security threats analysis should be carried out"* [Int10]
- *"If security threats have been identified, then a vulnerability analysis should be undertaken in order to specify security requirements"* [Int10]

In both the above mentioned text sections, the IEC 61508 standard references to the IEC 62443 series which covers industrial network and system security. However, the IEC 62443 series is not yet currently completely available. Parts of the series are completed and other parts are just available as drafts[1]. Still, safety certification organizations such as *TÜV Süd* or *exida* promote that they offer safety and security certification according to IEC 61508 and IEC 62443.

From the combined safety and security certification proposals it can be seen that there is a demand for considering security when certifying for IEC 61508. This demand has already been identified by the EWICS TC7 working group. This working group analyzes state of the art methods and evaluates their appropriateness for industrial use. The working group provides inputs for standardization committees and has also provided contributions to the IEC 61508 standard [Nor09]. It appears that through the input of such working groups, security becomes more relevant, including the IEC 61508 standard, even if just in the form of references to the IEC 62443 security standard.

[1]http://isa99.isa.org

ISO 26262

The ISO 26262 is a safety standard for the automotive industry, is based on the IEC 61508, and was first fully published in 2012. The standard itself does not cover security requirements, does not mention any security concerns, and does not even provide links to security-relevant standards. However, future technology trends such as car-to-car communication indicate that security issues will become more important in the automotive domain [IJ13]. Also [EPSW10] argues why security should be considered and proposes an integrated development process of safety according to ISO 26262 and security. The process consists of several joint phases for safety and security such as requirements engineering, but also some separate phases such as hazard analysis for safety and security risk analysis for security.

IEC 622xx

The IEC 62278, IEC 62279, IEC 62280, and IEC 62425 are relevant for safety-critical railway systems. In European countries equivalents to these standards, the EN 501xx series, exist. The railway standards are not targeted at security issues. For example, the IEC 62278 standard explicitly says: *"This International Standard does not specify requirements for ensuring system security"* [Int02].

However, other parts of the railway safety standard series do actually address security issues. The IEC 62280 standard addresses communication aspects of railway systems and considers closed as well as open communication systems which could be subject to security attacks. The standard focuses on safety and security related communication aspects. For example, one safety-related aim is to ensure message integrity. An additional security-related aim is to ensure its authenticity. Thus, the standard describes cryptographic measures like cryptographic signatures which achieve the aforementioned aims. However, the standard does not cover security aims less related to safety aims such as confidentiality. The railway standard series also only addresses security regarding railway communication systems and not for other parts of railway systems [Int02].

Several research papers report practical examples of systems in the railway domain being certified which also cover security in a more rigorous way than demanded from the standard [SL13]. [SSS09] analyzes the railway standards for shortcomings and amongst other measures suggests the use of an iterative risk reduction approach based on the railway standard to address safety and security aspects. Furthermore, a report of the MODsafe FP7 project (Modular Urban Transport Safety and Security Analysis) [Ber10] discusses security for the railway standards and suggests management approaches to incorporate security into the railway domain. The report also defines security responsibilities and liabilities.

IEC 61513

The IEC 61513 standard targets nuclear power plants. It is was developed by the SC45A subcommittee of IEC and is the top-level document for a series of nuclear power plant standards. It describes general requirements for systems and equipment used in nuclear power plants and, also to some extent, covers security activities (for example, the IEC 61513 standard explicitly requires security plans) [Int14b].

In 2008, the SC45A started to work on a series of documents targeting cybersecurity. The first and currently only published document of this series is IEC 62645. IEC 62645 addresses the software and system development process and extends, and at some points refines, security requirements of other nuclear power plant standards like IEC 61513. IEC 62645 defines three different levels of security and the rigor of the security requirements depends on the security level. The document describes a security lifecycle, partly based on the IEC 27000 series (which covers IT security) and is partly specific for the nuclear domain. Regarding the interplay between safety and security, a related document, IEC 62859, has been developed. IEC 62859 targets interactions between safety and security and gives recommendations for the coordination of the design and operation in the nuclear domain with respect to safety and security [PCQH13].

Based on the IEC 61513 and IEC 62645, [SBMS10] presents an overall development process specifically for the nuclear domain and focuses on security. One major step of the process is to develop and to verify a security plan covering management, operational, and technical aspects.

IEC 60601

IEC 60601 is a series of standards addressing medical devices. The current, third, version of this standard was published in 2010. The IEC 60601-1 standard defines general requirements for medical devices and links to other standards in the series for more specific details like, e.g. for software-specific requirements. The IEC 60601 series requires the usage of an ISO 14971 compliant risk analysis. This introduces some level of security as well, but is not specifically targeted at security [Int14a].

The IEC SC62A working group published the IEC 80001-1 standard in 2010. This standard describes information security management specifically targeted at IT networks with integrated medical devices and defines who is responsible for what. This standard is the most well known security related guidance for medical products, however it mainly addresses security aspects for medical device communication, and not for the devices themselves [MME^{+}13].

The IEC 27799 standard covers security requirements for medical devices, but it does not require or suggest security-relevant processes or lifecycles. Such a lifecycle is proposed in [RLRP13]. It integrates abuse cases, security requirements, risk analysis, and security tests into the development lifecycle of medical devices.

IEC 62061

The IEC 62061 targets the safety of manufacturing and machinery with particular focus on their control systems. The current version (1.1) was published in 2012. The standard itself does not cover security, because the domain rather targets mechanical and electromechanical parts. However, if interconnection of machinery devices give rise to security threats, the IEC 62443 standard series for industrial communication network security can be taken as a security guideline [Int14c].

IEC 61511

The IEC 61511 series targets the process industry and consists of three parts (IEC 61511-1, IEC 61511-2, IEC 61511-3). The standards are relevant, for example, for industrial plants such as chemical production, oil refineries, or power plants (but not nuclear power plants). The standard series was published in 2003 and the current version does not address security concerns. However, the second edition of the standard series is currently under development and it will also target security issues. For example, IEC 61511-2 will require a risk analysis from a security point of view [Fan13].

Specifically for power plants, the IEC TR 27019 technical report covers physical or network security issues and gives security guidelines with focus on the energy sector. For process industry in general, [OPF13] proposes an integrated safety and security hazard analysis and management approach. The hazards are gathered with a modified HAZOP analysis which also covers security aspects.

2.1.3 Security in other Safety Standards

This section covers security aspects of widespread safety standards which are not directly derived from the IEC 61508.

DO-178C

The DO-178C standard is relevant for the avionics domain. In Europe it is known under the name ED-12C. The standard focuses on topics such as testing or traceability of a system, however, it does not directly address safety [Bro08]. The recently published version (DO-178C) replaces the DO-178B standard, which was the relevant avionics standard for about 20 years [RTC12].

DO-178 does not explicitly mention security, however, the US-American SC-216 and the European WG-72 working groups work on security-related guidelines for avionic systems. So far, the DO-326/ED-202 guideline is published and describes security process specifications. Currently the working groups are developing additional security method and guidance documents.

Apart from the working groups, several researchers suggest combining DO-178 with Common Criteria (CC) security certification. [BBD$^+$12] suggests an integrated development process in accordance with DO-178 and CC. The proposed development process consists of different process steps and artifacts and links artifacts which share a common context (e.g. security objectives for CC and system requirements for DO-178). Also [TAFR02] suggests merging the two certification processes and maps CC requirements to the DO-178 process in order to show their similarities.

MIL-STD-882

The MIL-STD-882 is a safety standard for military devices and is widely spread in the USA. The first version of the standard was published in 1996 and the current version (MIL-STD-882E) was published in 2012. Unlike the IEC standards, the MIL-STD-882 is publicly available for free. The MIL-STD-882 standard does not cover security, but just mentions that security and privacy requirements have to be included in a system specification. However, no further guidance on security is given. The MIL-STD-882 applies to military devices in the US and for such devices the US Department of Defense provides other guidelines on security; however, no security standard and no integrated approaches regarding safety and security are provided [US 12].

2.1.4 Standards and Reports on Integrated Safety and Security

There is no internationally accepted standard which fully addresses security as well as safety. This section presents mature drafts and reports which target this area.

The SeSa Method

The SeSa Method was developed by the Norwegian research organization SINTEF and targets systems relevant to IEC 61508 and IEC 61511. The SeSa method covers the problem that many safety-related systems have to provide remote access to be feasibly maintainable. The aim of the SeSa method is to provide such a secure remote access, while still not greatly interfering with safety and in particular with safety certification. For example, one explicit aim is that *"no significant impact on the SIL level is revealed by use of the SeSa method"* [GJØO10].

SeSa models the impact of security on the safety system. The SeSa report discusses the relation between security certification levels and safety certification levels and argues that, if the security requirements are well specified and analyzed for the safety system, the safety certification process will be sufficient to ensure the correct functionality of the security-related parts of the safety system.

Figure 2.2 shows an overview of the SeSa method. Basically the method starts with a safety system and with a possible approach to access it remotely (e.g. VPN tunnel). Based on that system, SeSa finds security threats for the safety system with a modified HAZOP analysis that also considers security aspects. The analysis results in relevant threats. If the threats could affect the safety system, then countermeasures have to be taken and the system is analyzed again. SeSa provides guidance on such measures based on other security-relevant standards and guidelines such as the *BSI Baseline Protection Manual* or the *NISCC SCADA Good Practice Guide*. If, after the second analysis, the safety would still be affected, then the current approach used to access the system remotely has to be discarded and another remote access approach has to be investigated.

SafSec Methodology

SafSec (Safety and Security) is a project targeted at supporting combined safety and security certification, particularly for avionics systems. An aim of SafSec is to provide a framework that makes modular certification possible. Figure 2.3 shows an overview of the SafSec methodology which consists of the following main parts:

Combined Safety and Security Risks. One of the core elements of SafSec is a unified risk modeling process integrating safety hazards and security threats. Based on the risks, combined mitigation measures can be considered and therefore, more applicable or efficient solutions can be found.

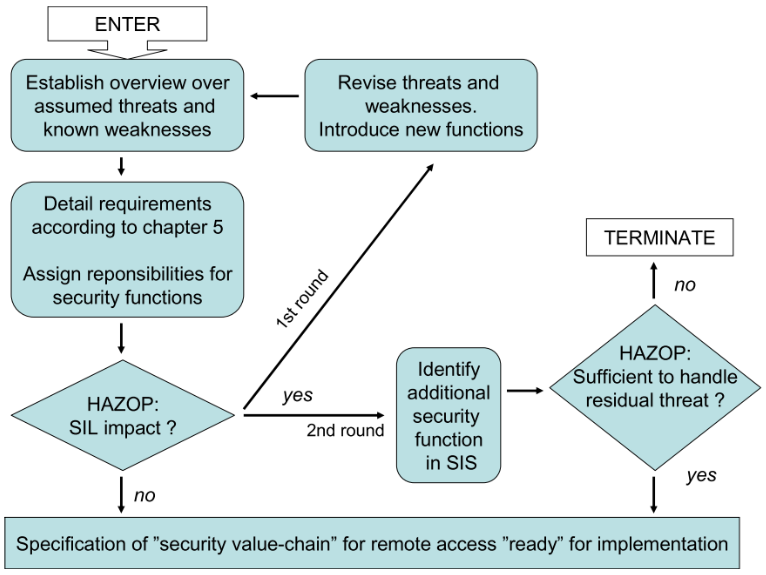

ENTER
Establish overview over assumed threats and known weaknesses
Revise threats and weaknesses. Introduce new functions
Detail requirements according to chapter 5
Assign reponsibilities for security functions
1st round
TERMINATE
no
HAZOP: SIL impact ?
yes
2nd round
Identify additional security function in SIS
HAZOP: Sufficient to handle residual threat ?
yes
no
Specification of "security value-chain" for remote access "ready" for implementation

Figure 2.2: The SeSa Method. This method is used to analyze the security of remote-accessible safety-systems. [GJØO10]

Risk Mitigation Design Process. The outcome of the design process should be documentation containing justification of which risks can be ignored and which measures are taken to mitigate other risks. The architectural model containing the risk reduction measures and the risk model serve as input to define a dependability specifications for all system components. These specifications are the basis for building structured dependability arguments which in the case of SafSec are realized as safety and security GSN diagrams.

Modular Certification. To make modular certification and reuse of certification artifacts possible, SafSec groups the GSN arguments which interact with each other into logically separable dependability arguments. The resulting components including their GSN arguments are called modules. The standard poses several requirements regarding these modules like, for example, that each module has to be analyzed regarding dependencies to other modules and that module boundaries have to be defined and checked. Based on these modules, a complete system including a complete GSN diagram describing the system's safety and security can be built.

A draft of the SafSec standard [DL06] was published in 2006. SafSec does not, like most safety standards, require specific processes to be followed, but it specifies particular targets and objectives that have to be achieved (e.g. verification that requirements are met according to a specified assurance level). Additionally, it gives guidance on how to achieve the targets. SafSec is actually a complete standard for safety and security, but so far it is not common practice in industry and thus it is not very influential [LCJ05].

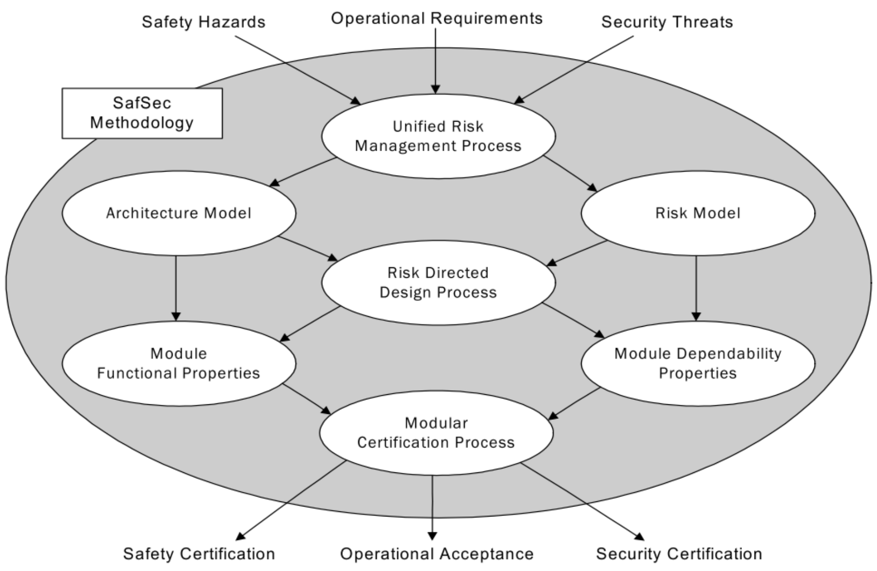


Figure 2.3: The SafSec Methodology. The main aims are to achieve combined safety and security certification as well as modular certification. [LCJ05]

SQUALE Project

The SQUALE project was conducted as part of a European FP4 program. The aim of the SQUALE project is to find a harmonized and unified approach to safety and security. SQUALE analyzes and combines approaches from both domains by first investigating which methods and processes are currently used in safety and security standards and then proposing a process which could be applicable for both. SQUALE does not aim to provide a generic standard of its own, but it describes an approach and criteria to develop products which can easily be certified to more specific standards, when developed with the SQUALE approach.

Figure 2.4 shows the overall development process of SQUALE. It is partly based on a predecessor of the CC security standard as it contains a *Dependability Target* (CC uses a similar concept of a Security Target). The Dependability Target describes a product which has to be certified and includes the necessary documentation that needs to be produced for an assessment. Complex systems can be split into several dependability targets, to make the certification process easier. However, in that case, SQALE demands strong arguments regarding the independence of the quality attributes of different Dependability Targets when being combined. The following steps are part of the development of a Dependability Target and have to be documented for assessment:

Dependability Objectives Definition. Based on the results of the hazard analysis, appropriate objectives (safety, confidentiality, availability, ...) have to be selected. Additionally, depending on the level of risk, integrity levels for the quality attributes have to be defined. These levels will define the rigor for the verification, validation, and quality assurance approaches.

Dependability Policy Definition. Implementation-independent high level measures have

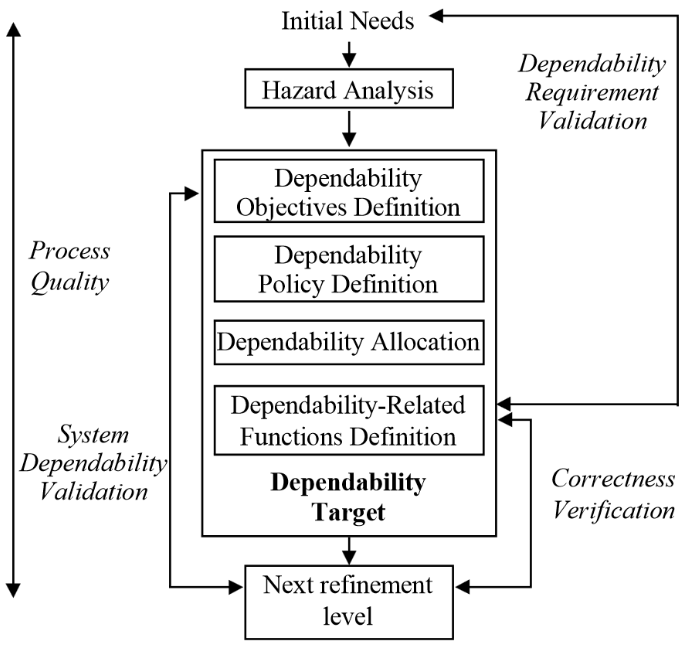

Initial Needs
Hazard Analysis
Dependability Requirement Validation
Dependability Objectives Definition
Dependability Policy Definition
Dependability Allocation
Dependability-Related Functions Definition
Dependability Target
Process Quality
System Dependability Validation
Correctness Verification
Next refinement level

Figure 2.4: The SQUALE Evalution Framework. The boxes describe activities during this process and the arrows represent quality assuring activities. [Cor99]

to be described. These measures describe how to cope with the defined objectives. Examples for such measures are design diversity or partitioning.

Dependability Allocation. This step defines which component of the system is responsible for which dependability objective by applying the above defined policy. The allocation is not just limited to software or hardware components, but could also include humans.

Dependability-Related Functions Definition. This step defines more specific functions that have to be implemented by the components to achieve the objectives via the policies.

Apart from the description of the dependability target, SQUALE requires that evidence be produced to prove that the safety and security goals are actually achieved. Such evidence is provided to an assessor in the form of documentation for four different quality assurance processes (the four arrows in Figure 2.4). These processes target verification, validation, and quality assurance for the Dependability Target. The rigor required for the processes depends on the integrity level for the components of the Dependability Target. For each of the four integrity levels, SQUALE provides process criteria for all quality assurance processes [Cor99].

The SQUALE draft [SQU99] was published in 1999 and was given to several organizations who develop standards such as ISO SC27, CENELEC, EWICS. SQUALE received good feedback and part of it might have influenced the development of current standard documents. However, SQUALE itself is currently rather outdated and not directly applied in industry.

2.1.5 Discussion

Currently, there is a gap between the level of security that a state-of-the-art safety device should provide, and the level of security that is demanded from such a device by safety standards. This limited awareness and consideration of security concerns in current safety standards on the one hand implies that there are no strict security requirements for such systems, and on the other hand implies that also there is no guidance on how to build safe and secure systems that are acknowledged by the safety standards. This leaves developers for safety-critical systems in a situation where on the one hand security is needed, but on the other hand they have to evaluate for themselves how and how much security to build in.

However, recent activities such as the IEC 62443 standard, the IEC 27000 series, or working groups like SC45A or EWICS TC7 which promote cybersecurity, give the impression that security concerns will be a relevant topic not just for future safety systems, but also for future safety standards.

2.2 Safety Patterns

This section first gives an introduction to design patterns and related concepts such as architectural tactics in general and then focuses on safety patterns. Safety patterns from literature and pattern-based development methods for safety-critical systems are presented.

2.2.1 Introduction to Patterns and Tactics

Patterns

A pattern presents a good and mature solution to a recurring problem. The concept of presenting solutions as a pattern can be traced back to Christopher Alexander. In his book *The Timeless Way of Building* [Ale79] he describes patterns for how to construct cities and buildings. The idea of writing well-proven solutions in pattern form spread to the domain of software engineering where it became way more famous than in architecture. In 1994, Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides published the book *Design Patterns - Elements of Reusable Object-Oriented Software* [GHJV94]. This book contains 23 design patterns for object-oriented programming languages. It became well-known in the software domain and from that time on, the concept of patterns became more and more applied to software engineering and is today part of best practice. The following is a definition of patterns tailored to the software domain:

> *"A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate."* [BMRS96]

Nowadays there are several different definitions of what a pattern is and also there are over a thousand different patterns for specific domains of software engineering as well as for other domains.

Architectural Tactics

Related to design decisions, though somewhat different to patterns is the concept of architectural tactics. The term was brought up by the Software Engineering Institute of the University of Carnegie Mellon in the book *Software Architecture in Practice* [BCK03] by Len Bass, Paul Clements, and Rick Kazman. They describe sets of general architectural design decisions and call them *Architectural Tactics.* These tactics describe architecture changes which affect single quality attributes like availability, or security. An example for a security tactic is *Limit Exposure* which is described as follows:

> *"Attacks typically depend on exploiting a single weakness to attack all data and services on a host. The architect can design the allocation of services to hosts so that limited services are available on each host."* [BCK03]

This description of the security tactic gives only a vague idea of how to apply the tactic. It does not give a precise solution within a precise context, such as patterns do. It rather describes a general concept which can and should be applied in most situations. There are different opinions and approaches to how to use tactics and on what they are. Some see tactics as not very well elaborated patterns, whereas others argue that most patterns can be related to tactic compositions [KP10a]. In any case, tactics are a design concept which provides quite a comprehensive overview of possibilities to improve specific quality attributes. For example, in [Har11], architectural tactics are used to improve quality attributes of architectural patterns.

Related to safety, Weihang Wu from the University of York presents a set of safety tactics [Wu03]. These safety tactics will be used throughout this thesis to characterize and organize safety patterns from literature.

2.2.2 Safety Patterns from Literature

The concept of patterns is nowadays widespread. There are thousands of patterns available, some of them for very specific domains. Not surprisingly, there are also several patterns described in literature that relate to safety. Table 2.1 gives an overview of current relevant literature describing safety-related patterns.

2.2.3 Pattern-Based Safety Development

The method described in this thesis is a pattern-based safety development method. There are already related methods described in literature. This section only describes these methods in brief. A detailed introduction and comparison of pattern-based safety development methods is given in **Publication 5:** *"Pattern-Based Safety Development Methods: Overview and Comparison"* which can be found in Chapter 7. The following describes these methods:

Applying Patterns along the IEC 61508 Safety Lifecycle. In [RVK13b], the authors suggest applying patterns during safety development. The authors analyze different phases of the IEC 61508 safety development lifecycle and provide links to different sets of safety patterns. They provide links to their own work on architectural safety patterns and also links to other research groups describing IEC 61508 process patterns.

Title	Description
[DKV97] *"The Reliable Hybrid Pattern - A Generalized Software Fault Tolerant Design Pattern"*	The paper describes a pattern which includes software fault tolerance techniques (e.g. N-version programming, voting, acceptance test). The pattern itself is presented in the form of a generic architecture. This architecture explicitly presents alternatives or variation points of the pattern (e.g. the usage of voting instead of an acceptance test).
[Dou98] *"Safety-Critical Systems Design"*	The article covers safety architecture patterns and discusses how they can be implemented. Most of the patterns are based on redundancy. An example is a pattern describing duplication of a channel.
[Sar02] *"A System of Patterns for Fault Tolerance"*	This paper introduces several architectural fault-tolerance patterns such as patterns for active and passive backup devices. Furthermore, the paper discusses how to group these patterns.
[Dou02] *"Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems"*	Besides other patterns, this book covers safety-related architecture patterns and also includes the patterns from [Dou98].
[Gru03] *"Transformational Patterns for the Improvement of Safety Properties in Architectural Specification"*	This paper covers safety patterns, some related to the patterns from [Dou02], as architecture transformations to increase the overall system safety.
[Han07] *"Patterns for Fault Tolerant Software"*	The book describes a large set of fault-tolerance patterns and groups them in a pattern language. The patterns are grouped as error detection, error processing, error mitigation, fault treatment, and architectural patterns and within the groups, the relationships between the patterns are described.
[Dou10] *"Design Patterns for Embedded Systems in C"*	The book presents design patterns implemented in C. Part of the book describes safety-related patterns based on [Dou02].
[Arm10] *"Design Patterns for Safety-critical Embedded Systems"*	This PhD thesis collects existing safety patterns for embedded systems from literature (mostly patterns from [Dou02] and software fault tolerance techniques from [Pul01]). Furthermore, the thesis introduces new safety patterns and presents all patterns in a consistent notation.
[Ham12] *"Survey of Safety Architectural Patterns"*	This survey presents the application of the patterns from [Arm10] within a company. The survey shows which patterns are applied more commonly and which are out of the ordinary. Furthermore, some new and rather domain-specific safety patterns are introduced.
[RVK12] *"Architectural Patterns for Functional Safety"*	The paper presents 4 patterns related to separating the safety functionality from non-critical system functionality.
[RVK13a] *"Patterns for Safety and Control System Cooperation"*	The paper presents 3 safety patterns that show how to efficiently integrate safety functionality into control systems.
[RK13] *"Patterns for Controlling System Safety"*	The paper presents 4 safety patterns related to the control systems domain.

Table 2.1: Overview of literature which introduces safety-related or fault tolerance patterns (Table based on **Publication 2**)

Safe Control System Method. The method [HS13] provides safety patterns addressing requirement issues, architectural issues, and assurance issues. The patterns provide defined interfaces (e.g. safety documentation required as input for a pattern) and the method suggests the use of a graphical notation to visualize the applied patterns as icons and their dependencies as arrows between them. Thus, one gets a simple graphical figure showing the information flow starting from requirements analysis going via architectural measures to safety assurance.

TERESA Project. The TERESA approach is a European FP7 project and provides a framework enabling model-based engineering for dependable embedded systems.

Several tools for modeling safety processes, pattern metamodels, and patterns themselves, as well as pattern repository tools are provided [HDGJ10].

SIRSEC Project. The aim of the SIRSEC project is to provide a platform for the development of safety-critical railway applications. SIRSEC works in collaboration with the TERESA project and also uses some of its tools. Additionally, SIRSEC provides an Eclipse/Papyrus tool for pattern integration [RHFP13].

Safety Tactic-Based Approach. The University of York put much work effort into the development and application of tactics for safety-critical systems. They provide a set of safety tactics and suggest a development process, which states anti-scenarios and mitigates them by the application of appropriate safety tactics [Wu07].

Safety Architecture Patterns. The method presents a collection of safety patterns in a PhD thesis [Arm10]. The patterns cover architecture decisions based mostly on redundancy (e.g. TRIPLE MODULAR REDUNDANCY pattern). A tool for selecting the patterns based on answers to questions like *"Is hardware redundancy possible"* is also provided.

Safety Architecture Patterns + UML. Antonio further investigates Armoush's safety patterns and captures them in UML representation. For these patterns he provides a tool which allows to easier integrate the patterns in a UML design [AKA12].

SDL Design Patterns. SDL (Specification and Description Language) is usually used to describe communicating systems. The authors of [FGG+05] formulate safety-related patterns (e.g. WATCHDOG) in UML and also add an SDL diagram to the patterns. These patterns are supposed to be used for safety-related systems already modeled in SDL.

REFLECT. The REFLECT (REndering FPGAs - Field-Programmable Gate Arrays - to MuLti-Core Embedded CompuTing) method presents a design flow of how to implement FPGAs. REFLECT provides its own safety requirement definition language and transforms an implemented system according to patterns, which are selected depending on the safety requirements, into another system which fulfills the safety requirements. For this, REFLECT uses patterns which contain annotations defining their interfaces for the transformation [PKCD11].

AltaRica Safety Patterns. The method [KSB+04] describes the formal application of redundancy-based architectural patterns. The patterns are described with linear temporal logic using the formal language *AltaRica*. The patterns can be integrated into AltaRica models and after the pattern application, attributes of the resulting architecture can be formally checked.

Safety Timing Templates. The method [Bit07] describes timing-related safety patterns and how to apply them. The patterns are held in a library and a user is asked questions in order to find appropriate patterns. The patterns formally describe a timing behavior and when integrated into an architecture, the timing behavior of the overall system can be formally checked.

2.2.4 Discussion

There has already been a lot of work done regarding safety patterns and there are numerous patterns available in literature. It is quite difficult to obtain an overview of current safety patterns; therefore, one aim of this thesis is to review current safety patterns and to organize parts of them. A further topic left open by current safety patterns is that most of them are limited to safety alone and do not consider security aspects.

Regarding security, there are some pattern-based safety development methods which already consider safety and security; however, none of the methods also structurally model safety and security aspects in the patterns or provide structured evidence to show that safety and security goals have been achieved.

2.3 Security Analysis of Patterns

This section covers patterns for which security aspects are investigated. Security patterns themselves will not be covered as they are not the focus of this thesis.

2.3.1 Security Analysis of Security Patterns

Halkidis et al. provide several papers on how to analyze patterns regarding their security attributes and on how to find a good match, including from a security point of view, when selecting a pattern. In [HCS04], they qualitatively analyze 13 security patterns regarding the security they achieve. The evaluation consists of three different parts:

- **Security Principles.** Halkidis et al. take security principles from Viega and McGraw [VM01] and informally discuss which of the principles are achieved or supported by the pattern. Examples for the security principles are *Defense in Depth*, *Principle of Least Priviledge*, or *Simplicity.*

- **Common Security Error Avoidance.** Regarding this aspect, Halklidis et al. discuss how well the security patterns help to withstand common security holes. The common security holes are also based on the book of Viega and McGraw [VM01] and examples are: *Buffer Overflows* or *Weak Access Control.*

- **STRIDE.** The last criteria against which the security patterns are evaluated are common threats. STRIDE is a threat analysis approach described by Howard and LeBlanc [HL03] and lists six types of common security threats: *Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service*, *Elevation of Priviledge.* Halkidis et al. discuss how well the security patterns counter these threats.

The outcome of the work of Halkidis et al. is a table which maps the security patterns to the security principles they follow, to the security holes they cover, and to the threats they counter.

Halkidis et al. present the application of the security patterns to a webstore case study in [HCS06a]. They analyze the case study for the three security aspects listed above and then decide which patterns to apply in order to adhere to the most security principles, to cover the most security holes, and to counter the most security threats.

Further work of Halkidis et al. shows how security patterns can counter security vulnerabilities [HCS06b]. They present an extended version of the webstore case study leaving several sources for attacks open. For example, in the case study, the initial implementation contains security holes like sources for SQL injection or sources for cross-size scripting which an attacker could exploit. They analyzed the security of the initial implementation by means of static security analysis tools and penetration tools. They also quantified the security risk of the system by investigating STRIDE threats and elaborating the risk of the different threats by means of fault trees. To get numeric values for the threat risks, they use fuzzy variables to describe the likelihood, exposure, and consequences of possible attacks. Next, they applied appropriate security patterns to counter such attacks. They chose the patterns according to the STRIDE threats they mitigate based on their previous work and for the resulting architecture they again analyzed the resulting STRIDE threats. The resulting analysis showed significantly lower security risk levels.

In [HTCS08], Halkidis et al. extend their work and put further focus on how to automate the security analysis approach. They implement the security patterns in UML and use annotations in the UML class names in order to identify specific parts later on. An example of such an annotation is that every class receiving some input data has to use the term 'Input' somewhere in the class name. Based on the UML diagram, they automatically make a STRIDE analysis and build a fault tree for the different STRIDE threats. For example, every 'Input' class is subject to the *Information Disclosure* threat and therefore information disclosure of this input would be added to the fault tree. With this fault tree given, the authors manually match security patterns which mitigate the high risk threats of the system. They show this approach for a case study and iteratively apply patterns to make the system more secure. After each iteration, the security fault trees are checked as to whether the threats are mitigated to a sufficient level. Otherwise more security patterns are manually applied.

2.3.2 Application of Security Analyzed Patterns

Yautsiukhin et al. propose a method of applying a structured security risk analysis based on the relevant threats for a system and on security patterns applied to a system [YS08]. They apply this method to an online publishing system case study.

- First, they gather general requirements for the system and map high-level security objectives to the requirements if applicable. For example, one requirement for the publishing system is that authors should be able to access their past papers. The corresponding high-level security objective would be *availability*.

- Next, STRIDE threats are mapped to the high-level objectives and for each STRIDE threat of the system, a threat tree containing possible attacks is constructed. For the attacks they evaluate the risk by estimating the damage potential, reproducibility, exploitability, affected users, and discoverability of attacks related to the STRIDE threats.

- They provide calculation rules to propagate the risks from the attacks up to the threats and security objectives. Furthermore, they rank the importance of the requirements in order to obtain a single number that represents the security of the system.

- They manually apply security patterns to see how the overall system security changes. Just as with the approach of Halkidis et al., the patterns contain information about how well they help to mitigate STRIDE threats. Thus, the overall security risk reduction for the system can be calculated.

The authors of [SB12] do not cover patterns, but still they provide an automated tool-based approach to conduct a STRIDE analysis on architecture block diagrams. Such an approach could also be applied to patterns and the work of Yautsiukhin et al. and Halkidis et al. described above could benefit from this approach.

2.3.3 Discussion

There are only a view structured security analysis methods available. In particular, there are scarcely any when it comes down to modeling the security of an early-stage architecture of a system without already modeling specific attacks and modeling the related probabilities and risks. As can be seen from literature, when introducing security aspects to patterns, STRIDE appears to be applicable. Therefore, STRIDE will be used to analyze security aspects of the safety patterns presented in this thesis.

2.4 Structuring Evidence for Safety Compliance

This section covers methods used to analyze and model safety systems and methods to produce evidence that such a system is safe. The section particularly focuses on methods which have also been applied or adapted to the security domain and which would be applicable when modeling security aspects into safety patterns.

2.4.1 Safety Case

A safety case is a structured argument indicating that a system or part of the system is safe and according to [NVSB12] is the most commonly used method for safety assurance. The aim of a safety case is to prove a claim by connecting evidence via arguments. A safety case can be defined as follows:

> *"A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context."* [Mel13]

The general structure of a safety case is first described in [BB98]. The notation is shown in Figure 2.5 and consists of the following main elements:

- *Claim.* A claim about the system or a property is made and has to be proven.

- *Evidence.* Evidence is used to prove the claim. Evidence can be scientific facts, test reports, documentation, but also assumptions or sub-claims.

- *Argument.* The argument links the evidence to the claim and consists of interference rules.

- *Interference.* Interference rules build up the argument. For example, if the argument is related to probabilities of failure, then the interference rules would be the probabilistic calculations from the evidence to the claim.

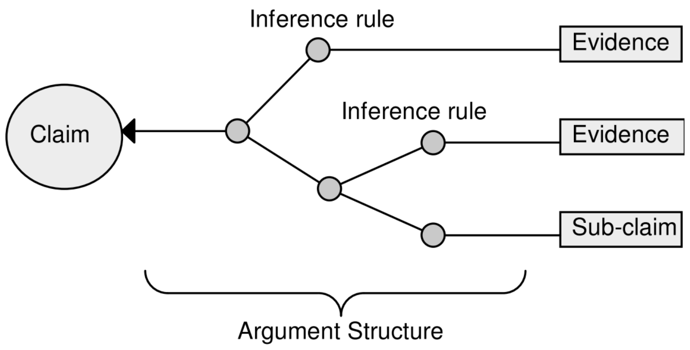


Figure 2.5: Claim Argument Evidence Notation. The figure shows how a generic claim is fulfilled by evidence elements which are connected to the claim via arguments. [BB98]

Safety cases can be made in simple textual representation, recently however graphical notations of safety cases have become more common. The following are the two most commonly used graphical safety case notations which are described in the following: **Goal Structuring Notation (GSN)** and **Claim Argument Evidence (CAE)**.

Claim Argument Evidence

CAE was developed by the company Adelard which also provides a safety case editor supporting CAE. The key elements of CAE are shown in Figure 2.6 and they are Claims, Arguments, and Evidence. There are no reports regarding the usage of CAE for the security domain, however in the safety domain CAE is used due to its tool support in the Adelard safety case editor.

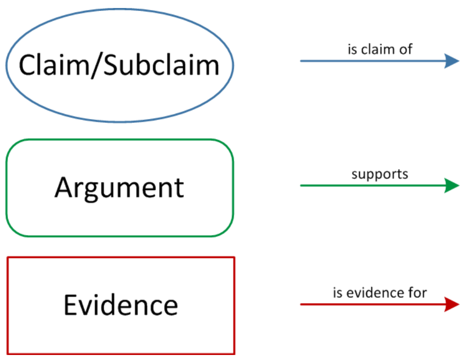


Figure 2.6: Claim Argument Evidence Notation. The figure shows the main elements and their notation for building a CAE diagram.

Security aspects of safety systems are inspected by extending a CAE safety diagram in [BNS13]. The authors discuss how to bring security into the system and analyze each safety claim of an example system. For each claim they thoroughly discuss how security could affect it and, if necessary, they add additional security sub-claims.

Goal Structuring Notation

The Goal Structuring Notation is widely used in the safety domain and several reports and scientific papers also report its usage for security systems. GSN is more precisely defined compared to CAE and since 2011 there is even a standard for the GSN notation available [GSN11]. The standard gives an introduction to safety cases, defines the structure and the elements of GSN, and also provides guidance on how to apply GSN. Figure 2.7 shows the basic elements used in GSN diagrams.

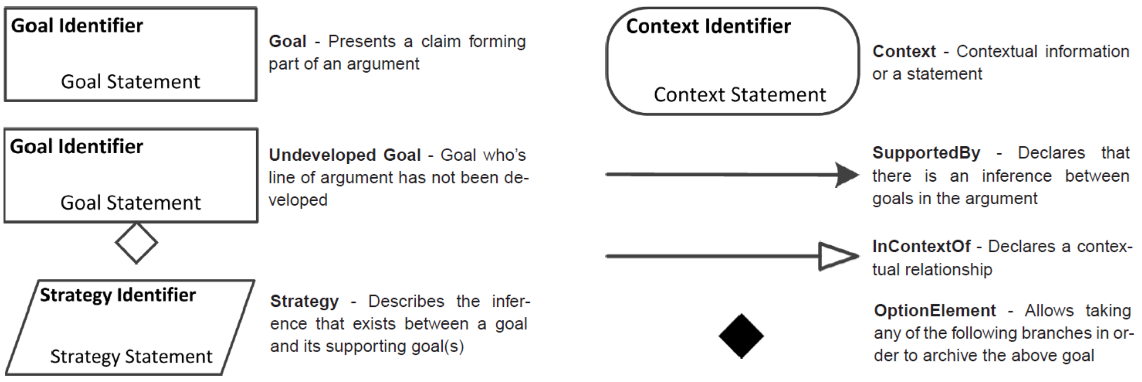

Goal Identifier
Goal Statement
Goal - Presents a claim forming part of an argument
Context Identifier
Context Statement
Context - Contextual information or a statement
Goal Identifier
Goal Statement
Undeveloped Goal - Goal who's line of argument has not been developed
SupportedBy - Declares that there is an inference between goals in the argument
InContextOf - Declares a contextual relationship
Strategy Identifier
Strategy Statement
Strategy - Describes the inference that exists between a goal and its supporting goal(s)
OptionElement - Allows taking any of the following branches in order to archive the above goal

Figure 2.7: Goal Structuring Notation. The figure shows the main elements and their notation for building a GSN diagram.

GSN Patterns. For GSN there is extended literature available for case studies when it is applied and there is even literature on GSN patterns like [KM98]. GSN patterns are more or less GSN diagram templates which can be used quite generically and refined for a specific safety argument. An example for a GSN pattern is the As-Low-As-Reasonably-Practicable pattern shown in [KM98]. It consists of a GSN diagram describing the main idea which is to reduce as many risk factors as feasibly possible and it contains further descriptions, like the consequences of applying the GSN pattern.

GSN for Security. The following shows selected work on security modeled with GSN:

- Using GSN for safety and security has already been suggested by the SafSec standard (see Section 2.1.4). The authors of [CL07] further explain the approach of SafSec and provide generic GSN templates which can be used to build safety or security arguments in general. The GSNs contain general goals like *"Identify threats"* or *"Security mitigations for threats"*. Based on those generic goals, one has to develop a specific GSN. The safety and security GSNs are separated in this approach.
- Security cases with GSN are covered in [GLW12]. They describe security cases in general and give guidance on how to construct a security case. For example,

they show how to build a security case assuring that a system is secure against buffer overflow attacks. Furthermore, they show how such a security case can be transformed into a security case pattern in order to be reused.

- The integration of security aspects in safety GSN diagrams is shown in [JY11a] and [Joh11]. The safety of a satellite system is modeled in GSN and in addition to the system, security aspects related to ‘insider threats’ are considered and integrated into the safety GSN. Every safety goal was simply analyzed regarding the additional threat and if the goal is subject to attacks, the security aspects were additionally modeled as subgoals. In [JY11b] the security of the same system is modeled with a different approach: *Boolean logic Driven Markov Processes.*

2.4.2 Boolean logic Driven Markov Processes

In [PCB09], Ludovic Piètre-Cambacédès et al. suggest the usage of Boolean logic Driven Markov Processes (BDMP) instead of regular attack trees for security modeling. BDMP has in the past only been used in reliability engineering and provides its strength in the modeling of probabilities and processes. Compared to regular fault trees, BDMP provide an additional notation called ‘trigger’ which allows the modeling of sequences and dependencies, thus, events in the tree can be combined. The BDMP provides probabilities that such a trigger leads to the event the trigger is pointing to.

Figure 2.8 shows an example for a BDMP security model. The model contains triggers which can be equipped with probabilities. The BDMP notation is quite similar to classical fault or attack trees and is thus expected to be easily understandable to safety as well as security experts. Based on the BDMP model containing the probabilities, one can compute all paths leading to the top-level event. These paths are then ordered regarding their probability to provide a good overview of the most critical parts of the system from a safety and a security point of view [PCB10b].

2.4.3 Fault Trees

Fault trees are a well known technique for the analysis of safety-critical systems. Fault trees contain probabilities of their leaf events and they connect the leaf events via logical relations (AND, OR, ...). Thus, the leaf probabilities can be propagated to the root of the tree to obtain an overall probability describing the root which usually represents some kind of safety goal.

In [NMD09] the authors present fault trees and attack trees with their mathematical definitions. Attack trees are similar to fault trees, but they usually describe security objectives as root and break them down to attacks which could break the objective when exploited. The authors argue that attack trees can easily be integrated into fault trees, if the root element of an attack tree is the leaf element of a fault tree. They provide mathematical formulas to propagate the probabilities of an attack tree to the fault tree.

Another security-related approach to fault trees is described in [SLS$^+$13]. The authors use component fault trees. They are regular fault trees with additional information to build components out of the fault tree. Such components can then be used as part of another fault tree. This makes it easier to mange huge fault trees. Every node in the

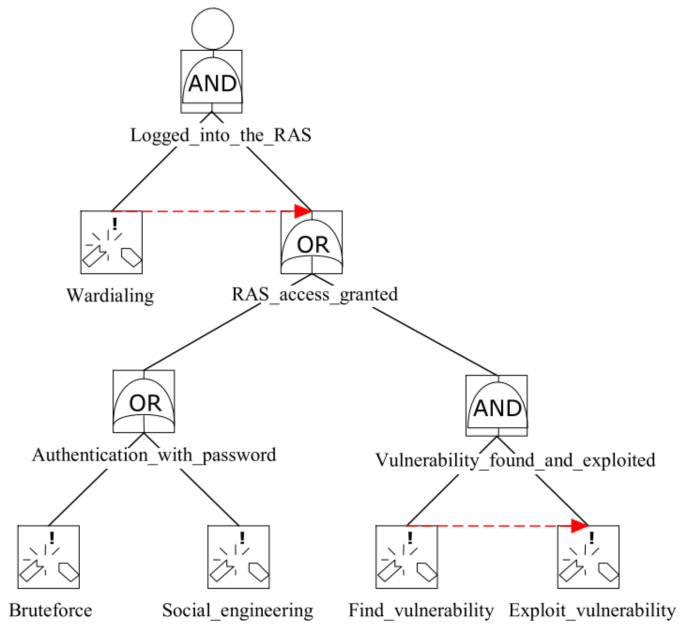


Figure 2.8: Boolean logic Driven Markov Processes. Example model for the security of a Remote Access Server system. The dashed lines represent the triggers. [PCB10a]

component fault tree is analyzed regarding its STRIDE threats. If one or more of the STRIDE threats are relevant, they are added as a sibling to the node, connected to it via an OR relation. For the security events, no probabilities are assigned, but simply the values ‘low’, ‘medium’, or ‘high’ are assigned. Thus, the safety and security aspects are modeled in the same fault tree, but they are not propagated to a single probability value representing the safety and security of the system.

2.4.4 Discussion

The above discussed methodologies are the only ones which address safety and security and which are developed and supported by more than just a single researcher, but by at least a working group. From these methodologies, GSN will further be used in this thesis to model safety and security of patterns, because GSN is an approach which looks promising and is currently widespread in the safety domain. From the above described methodologies, GSN is the one which, according to research articles, is most commonly used in research as well as in industry.

2.5 Summary and Difference to the State-of-the-Art

The increasing number of research articles which address security aspects for safety systems show that there is a reaction to security-related safety incidents such as the Stuxnet attack. Several research communities promote methods and processes to model and improve the level of security for safety systems. Related to the pattern community, several pattern-based safety development methods such as the one presented in the TERESA project show that here also safety and security aspects as a whole have to be covered. However, apart from the TERESA project, which actually focuses more on security, there are no combined safety and security methods available and there are no detailed reports of industry applying pattern-based safety and security development methods. Thus, introducing security aspects to pattern-based safety development is a relevant and novel research topic.

Another recent topic for safety-critical systems is how to provide evidence that a system complies to safety standards. In particular, it is of interest how to provide arguments that security goals of a safety-critical system have been achieved. Current safety standards do not yet intensively cover security; however, recent development of safety standards and reports of several working groups indicate that security will become more important for safety systems. Thus it is of great interest to find an approach which provides guidance on how to develop safety and security critical systems and how to provide evidence that safety and security goals are achieved. It is of particular interest to provide such evidence which complies to safety standards, because then the evidence can be used as a helpful argument during safety certification.

To provide such a pattern-based development method considering safety as well as security, domain-specific patterns have to be provided. In literature there are patterns for safety systems and there are patterns for security systems; however, there are no patterns available which address both aspects in detail. Introducing new patterns addressing both would, on the one hand, be a lot of work and on the other hand would be a questionable approach, because there are already mature patterns in both domains available. Therefore, integrating security aspects into existing safety patterns appears to be feasible. There has been research addressing the topic of analyzing security aspects like threats and attacks of patterns, but that has not yet been applied to safety patterns.

To summarize, this thesis provides the following improvements to state-of-the-art:

- Safety patterns from literature are gathered, structured, and organized and security aspects of current safety patterns are analyzed. This results in a collection of high-level safety architecture patterns containing security aspects.
- Current safety patterns are enhanced with GSN models presenting security and safety arguments for the patterns. The GSN models contain explicit references to the IEC 61508 safety standard to provide a structured approach to argue for the safety of a system based on the actual standard.
- A pattern-based safety and security development method is suggested. In literature there is no other such method available which considers safety and security and which also produces evidence elements to be used for safety certification.

Chapter 3

Safety Patterns with Security Aspects

This chapter describes how to systematically construct a system of safety patterns including their security aspects based on safety patterns from literature.

Figure 3.1 shows an overview of the steps conducted to build such a pattern system. From a safety point of view, the safety patterns from literature are analyzed regarding the safety tactics they can be related to and a tree representation of these tactics is constructed. The tactic tree representations are used to analyze relations between the patterns and to construct a GSN safety diagram for the patterns. For the GSN safety diagram, additional safety relevant scenarios for the patterns serve as input. From a security point of view, the patterns are analyzed regarding their safety-relevant security threats and these threats are organized in a GSN diagram. The above mentioned steps to build the safety and security GSN diagrams and to find relations between the patterns are described in detail in this chapter and in the **Publications 2, 3,** and **4**.

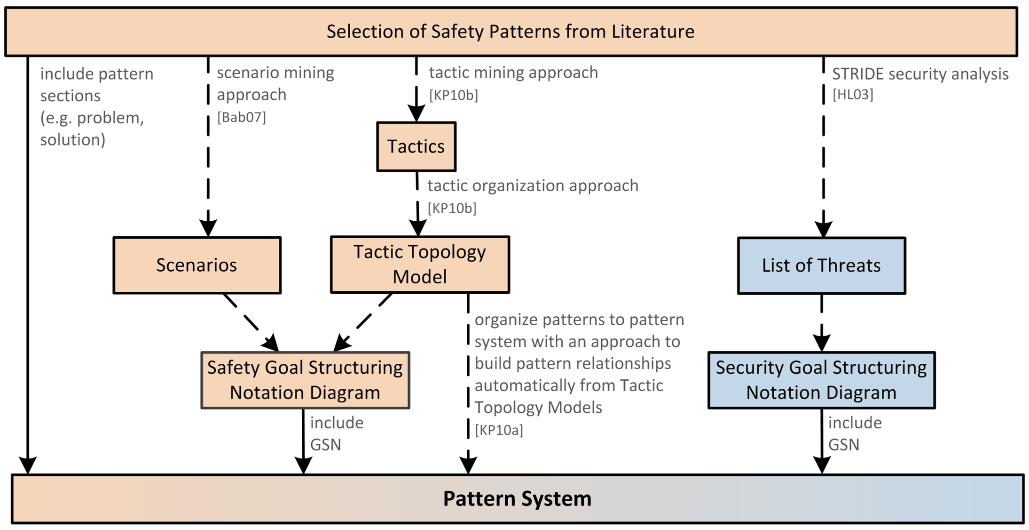

Selection of Safety Patterns from Literature
include pattern sections (e.g. problem, solution)
scenario mining approach [Bab07]
tactic mining approach [KP10b]
STRIDE security analysis [HL03]
Tactics
tactic organization approach [KP10b]
Scenarios
Tactic Topology Model
List of Threats
organize patterns to pattern system with an approach to build pattern relationships automatically from Tactic Topology Models [KP10a]
Safety Goal Structuring Notation Diagram
Security Goal Structuring Notation Diagram
include GSN
include GSN
Pattern System

Figure 3.1: Overview of steps described in this chapter to build a safety pattern system with security aspects from safety patterns in literature (based on **Publication 4**).

Throughout this chapter, an example pattern, the Homogenous Duplex Pattern, will be used to show how this safety pattern from literature is integrated into the safety pattern system with security aspects. The Homogenous Duplex Pattern basically describes a system using an active backup channel which it is switched to if a failure in the primary channel occurs. Figure 3.2 shows the structure of this pattern.

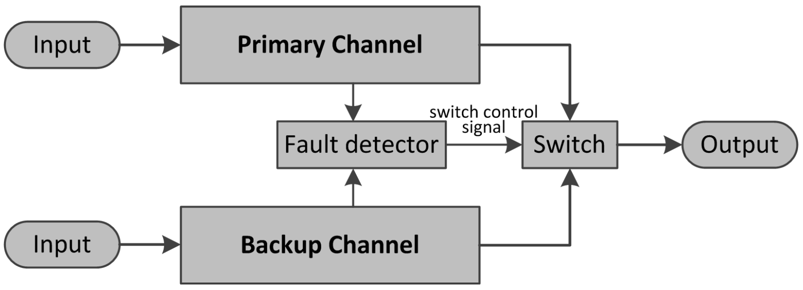


Figure 3.2: Graphical representation of the components from the Homogenous Duplex Pattern. If a primary channel fault is detected by the fault detector, the switch switches to the output of the backup channel (based on **Publication 2**).

The approach to develop the Homogenous Duplex Pattern described throughout in this chapter has been applied to several selected patterns from literature in order to build up a system of safety patterns with security aspects. The whole pattern system including all selected and developed safety patterns is presented in **Publication 4**.

3.1 Selection of Safety Patterns

Literature provides several safety patterns as already presented in Section 2.2.2. This thesis focuses on architectural safety patterns. These are patterns describing high-level architecture design decisions such as whether some components should be installed or developed redundantly or whether there should be a component such as a watchdog. Armoush [Arm10] already gives a comprehensive overview of such patterns from literature. Therefore, the patterns presented in this thesis are mostly based on the patterns from Armoush and some additional patterns are extracted from the IEC 61508 safety standard. The following list gives the names of the safety patterns covered. The full pattern descriptions can be found in **Publication 4**.

- Homogenous Duplex Pattern
- Heterogenous Duplex Pattern
- Triple Modular Redundancy Pattern
- M-out-of-N Pattern
- M-out-of-N-D Pattern
- N-Version Programming Pattern
- Acceptance Voting Pattern
- Recovery Block Pattern

- N-Self Checking Programming Pattern
- Sanity Check Pattern
- Monitor-Actuator Pattern
- Watchdog Pattern
- Safety Executive Pattern
- Protected Single Channel Pattern
- 3-Level Safety Monitoring Pattern

3.2 Selection of a Pattern Form

The pattern form used for the pattern system is based on the pattern form presented in [Bab07]. This targets architectural patterns with specific focus on presenting the pattern in a way that can aid architectural reasoning or architectural review processes. For example, the pattern form explicitly covers architectural information of the pattern such as scenarios and affected quality attributes. The following presents the different sections of the chosen pattern form.

- **Pattern Name.** A representative name of the pattern mostly based on the name suggested in [Arm10]
- **Pattern Type.** Classification into hardware/software and fail-safe/fail-over
- **Also Known As.** Other names for the pattern used in literature
- **Context.** Situation giving rise to a problem
- **Problem.** The problem the application of the pattern solves
- **Forces.** Reasons why the problem is hard to solve
- **Solution.** The proven solution to the problem. The solution contains a graphical description of the components involved.
- **Safety GSN.** A GSN skeleton to be used for safety reasoning
- **Security GSN.** A GSN skeleton to be used for security reasoning
- **Consequences.** Benefits and liabilities of applying the pattern with focus on the effect on quality attributes
- **General Scenarios.** Safety-related scenarios which are used to build the safety GSN and which can be used for architectural reasoning
- **Known Uses.** Real application examples of the pattern
- **Credits.** References to previous work on the pattern

3.3 Safety GSN

The main aim of the activities described in this section is to construct a GSN diagram which can be used for safety reasoning when applying the pattern. To build the GSN diagrams, first tactics and scenarios are mined from the patterns and based on these tactics and scenarios, the GSN diagram is constructed. This section is based on **Publication 2**.

3.3.1 Scenario Mining

To aid safety reasoning of the patterns, safety-related scenarios are explicitly stated for the patterns. To mine the scenarios from existing patterns, an approach presented in [Bab07] is used. The approach is simply to manually search the problem and solution statements of a pattern for aims and goals of the pattern and to then describe these aims and goals in the form of a scenario. For example, part of the problem statement of the Homogenous Duplex Pattern is:

> *"Make the system continue operating in presence of a fault in one of the system components."* [Arm10]

When combining this with the solution of the pattern (which is to have two redundant channels), the following example scenarios can be stated:

- The system is fully operational even in the case of a single channel failure.
- A single channel random fault does not lead to a system failure.

3.3.2 Analyzing Safety Tactics for the Patterns

Tactics are not directly part of the pattern, however, the safety GSN skeletons are built based on tactics in the patterns and the patterns are organized according to their tactics as will be described in the following sections.

Kumar et. al [KP10b] present a structured approach to analyzing object-oriented design patterns regarding the general design decisions (architectural tactics) these patterns apply. Kumar et al. manually analyze the textual sections of patterns and look for keywords in each section which are also present in the description of any architectural tactic in order to find the tactics used by the pattern. This approach is also applied in this thesis, but to safety patterns instead of object-oriented patterns. For that, safety-relevant tactics are required which are initially proposed in [Wu03] and refined in **Publication 1**. The whole set of safety tactics is shown in Figure 3.3 and the detailed descriptions of the tactics can be found in **Publication 1**.

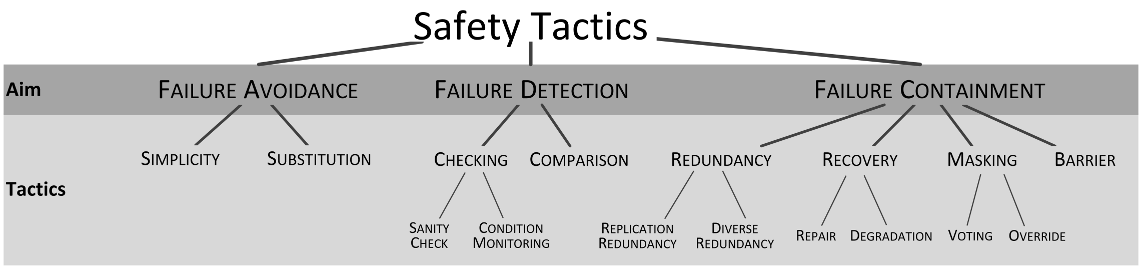


Figure 3.3: Architectural tactics for safety. The tactics are grouped into principles which avoid failures, which detect failures, and which contain failures (based on **Publication 1**).

To find the tactics of the Homogenous Duplex Pattern, its text sections have to be analyzed. For example, the following text shows part of the abstract section:

"It is a hardware pattern that is used to increase the safety and reliability of the system by providing a replication of the same module (Modular redundancy) to deal with the random faults." [Arm10]

According to the approach of Kumar et al., the keywords *replication* and *modular redundancy* indicate that *Replication Redundancy* (which is a safety tactic) is applied by this pattern. A full analysis of all text sections of the HOMOGENOUS DUPLEX PATTERN shows that the following safety tactics are applied by the pattern (the detailed analysis can be found in **Publication 2**):

- Replication Redundancy
- Override
- Condition Monitoring

Kumar et. al [KP10b] further describe how to structure the tactics found in the form of a diagram. They suggest finding the main goal of the pattern in the problem statement (which in the case of the HOMOGENOUS DUPLEX PATTERN is to *"Continue operation even in case of faults"*) and then connect the tactics which fulfill this goal by arrows. Additional information about what in particular a tactic achieves, or how it achieves it, is given in textual form next to these arrows. If a tactic gives rise to other goals addressed by another tactic in the pattern, then these tactics are again connected with arrows. Figure 3.4 shows the resulting tactic representation of the HOMOGENOUS DUPLEX PATTERN.

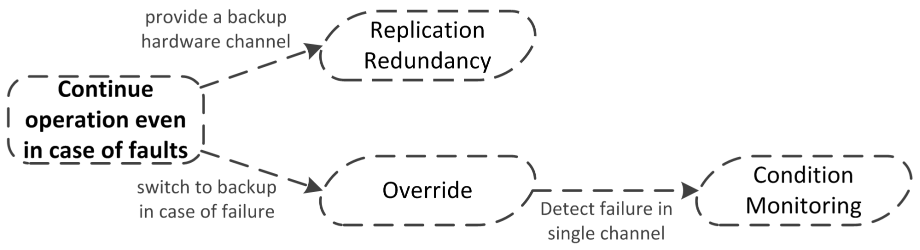

provide a backup hardware channel
Replication Redundancy
Continue operation even in case of faults
switch to backup in case of failure
Override
Detect failure in single channel
Condition Monitoring

Figure 3.4: Tactic representation of the HOMOGENOUS DUPLEX PATTERN (based on **Publication 2**).

3.3.3 Building the Safety GSN based on the Tactics and Scenarios

The safety GSN skeletons should be used to reason about the safety of the system after applying the pattern; therefore, all safety GSN skeletons start with the main goal that *"The system maintains its safety functionality"*. Based on the scenarios of the pattern, GSN subgoals are added. If the scenarios are independent from one another, they are simply added as subgoals to the main goal. If scenario A depends on scenario B, then B is modeled as subgoal of A. The tactics which help to achieve the scenarios are put as GSN strategy elements below the corresponding subgoals. If contextual information of the pattern is relevant, it is added as GSN context elements.

Figure 3.5 shows the constructed safety GSN skeleton for the HOMOGENOUS DUPLEX PATTERN. All the mined tactics and scenarios from the pattern are part of the GSN diagram as GSN strategies and GSN goals respectively. For the pattern, the scenario *"The system is fully operational even in case of a single channel failure"* depends on the scenario *"A single channel random fault does not lead to a system failure"* and therefore the later one is modeled as a subgoal of the first. The tactics (*Replication Redundancy*, *Override*, and *Condition Monitoring*) are directly modeled below the scenario which they primarily address. Additional undeveloped subgoals (e.g. "Common cause failures are sufficiently low") are added to the diagram to express additional safety-relevant assumptions for the pattern. These undeveloped goals have to be developed (i.e. an argument for achieving these goals has to be found) when implementing the pattern.

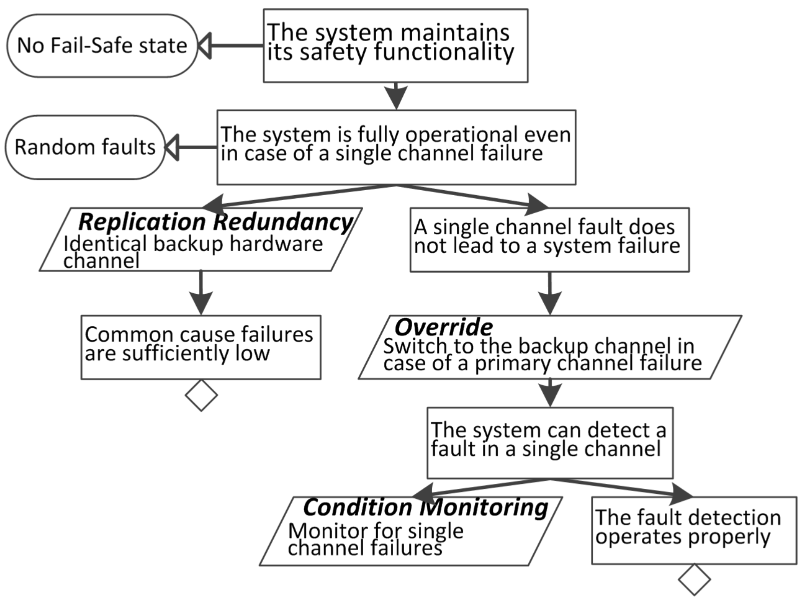

No Fail-Safe state
The system maintains its safety functionality
Random faults
The system is fully operational even in case of a single channel failure
Replication Redundancy
Identical backup hardware channel
A single channel fault does not lead to a system failure
Common cause failures are sufficiently low
Override
Switch to the backup channel in case of a primary channel failure
The system can detect a fault in a single channel
Condition Monitoring
Monitor for single channel failures
The fault detection operates properly

Figure 3.5: Safety GSN skeleton of the Homogenous Duplex Pattern (based on **Publication 4**).

3.4 Security Aspects of Safety Patterns

This section describes how to analyze the safety patterns regarding their safety-relevant security threats and how to construct GSN diagrams representing these security threats. The content of this section is based on **Publication 3**.

3.4.1 Threat Analysis

The safety patterns provide an architecture description on a very general level. Therefore, the chosen method for investigating security aspects of the safety patterns is a security threat analysis which is usually conducted during the early phases of security design.

STRIDE

The chosen threat analysis is the STRIDE threat analysis proposed and used by Microsoft [HL03]. The reason for choosing STRIDE is that it is the most well known type of threat analysis and even more importantly, it is well documented. STRIDE requires a data flow diagram as input. A data flow diagram describes inputs, outputs, and communication between elements of a system. STRIDE goes through all elements of such a diagram to list relevant threats. STRIDE describes the following six types of threats:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service (DoS)
- Elevation of Priviledge (EoP)

Not all of these threats are relevant for all types of elements in the data flow diagram. STRIDE provides information about the relevant threats that have to be considered (see Table 3.1). After going through the data flow diagram and listing relevant threats for the elements, one gets a list of threats which in later steps during the development lifecycle have to be rated and mitigated.

DFD element type	**S**	**T**	**R**	**I**	**D**	**E**
External entity	X		X			
Data flow		X		X	X	
Data store		X	X	X	X	
Process	X	X	X	X	X	X

Table 3.1: STRIDE threats mapped to the four different types of data flow diagram elements.

STRIDE Adaptation to be used for the Patterns

The STRIDE analysis has to be adapted to be applied to the patterns. The safety patterns contain diagrams quite similar to data flow diagrams. The patterns contain blocks describing processing elements (in hardware or in software) and they contain connections between the blocks which show the flow of data. The processing elements are similar to the *Process* entity of the STRIDE method and the connections are similar to the *Data flow* entity. However, not all STRIDE threats are primarily relevant for safety-critical systems. For example, *Repudiation* is not usually important for such systems. Also *Information Disclosure* does not directly affect the system in a way that could bring it into a critical state. Therefore, only a limited set of threats are considered when analyzing the pattern diagrams regarding the two chosen entities as shown in Table 3.2.

DFD element type	**S**	**T**	**R**	**I**	**D**	**E**
Data flow		X			X	
Processing element	X					X

Table 3.2: Considered STRIDE threats for the entities of the graphical diagrams which are part of the safety pattern descriptions.

Application of the Threat Analysis to a Pattern

Figure 3.2 (page 32) already showed the elements and their data flows of the Homogenous Duplex Pattern. The figure contains the following four main elements and their data flows:

- Primary Channel
- Backup Channel
- Fault Detector
- Switch

All these elements are analyzed regarding the relevant threats for *Processing elements* and their connections were analyzed regarding the relevant threats for *Data flows*. This results in the following list of threats for the pattern:

- Tampering of Single Channel input data
- DoS of Single Channel input data
- Spoofing of Single Channel
- EoP on Single Channel
- Tampering of Single Channel output data
- DoS of Single Channel output data
- Tampering of Fault Detector input data
- DoS of Fault Detector input data
- Spoofing of Fault Detector
- EoP on Fault Detector
- Tampering of Fault Detector output data
- DoS of Fault Detector output data
- Spoofing of Switch
- EoP on Switch
- Tampering of Switch output data
- DoS of Switch output data

This list of threats is not structured and thus it is not easy to see which of the threats are important and which are less important. For example, the *DoS of Fault Detector output data* threat does not affect the functionality of the system as long as the primary channel works properly. On the other hand, the *Tampering of Fault Detector output data* threat directly affects the system and can lead to critical states. Therefore, this threat is more important.

3.4.2 Building the Security GSN based on the Threats

To visualize which threats are very important and which threats only become important when combined with others, the threats are displayed in a GSN diagram (Figure 3.6). All security GSNs start with the main goal that *"Safety-critical functions are maintained in case of attack"*. Directly linked to that subgoal are goals related to threats which can directly lead the system to a critical state when exploited. The subgoals are formulated in the form "Threat X is prevented" and are represented with undeveloped GSN goals. This means that a developer has to think about and mitigate these threats when applying the pattern. From the remaining threats which are not yet included in the GSN diagram, any combination of threats which could lead the system into a critical state are also added to the diagram. These combinations are included by using the *GSN Option Element* (however, there are no such threats for the pattern in Figure 3.6). Any remaining threats only affect the non-critical system functionality and are thus not highly relevant for safety. Therefore, these threats are not included in the GSN diagram.

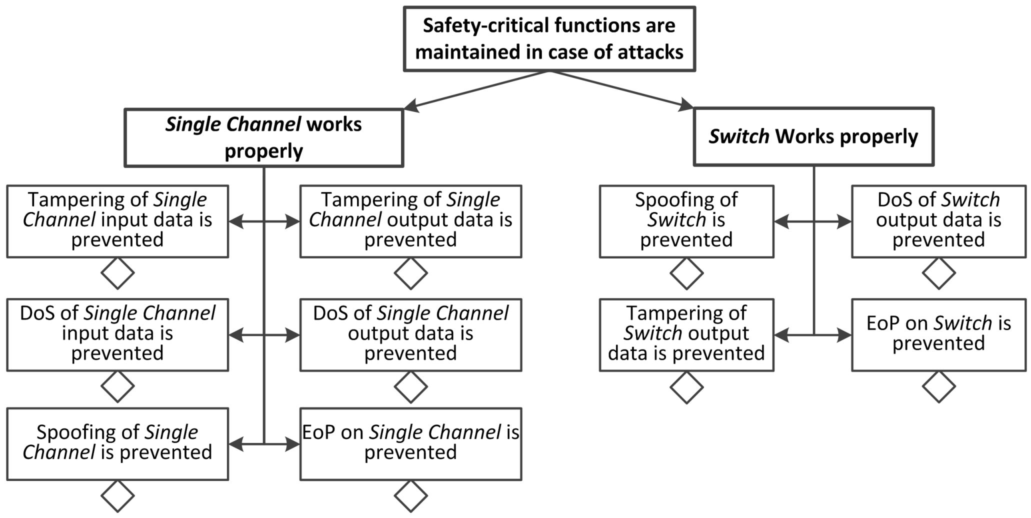

Safety-critical functions are maintained in case of attacks
Single Channel works properly
Switch Works properly
Tampering of *Single Channel* input data is prevented
Tampering of *Single Channel* output data is prevented
DoS of *Single Channel* input data is prevented
DoS of *Single Channel* output data is prevented
Spoofing of *Single Channel* is prevented
EoP on *Single Channel* is prevented
Spoofing of *Switch* is prevented
DoS of *Switch* output data is prevented
Tampering of *Switch* output data is prevented
EoP on *Switch* is prevented

Figure 3.6: Security GSN skeleton of the Homogenous Duplex Pattern (based on **Publication 3**). Not all, just safety-relevant threats are included. They are represented as undeveloped goals and have to be elaborated by a devleoper.

3.5 Organizing the Safety Patterns to a Pattern System

To organize the patterns and to find their relations, a structured approach presented in [KP10a] is used. The approach is to compare tactic representations (as it was already shown in Figure 3.4 on page 35) of the patterns in order to find their relations. The approach defines relations between two patterns such as *is similar* or *refines*. These relations are mapped to logical predicates describing attributes of the tactic representations of the two patterns. If a predicate matches, then the relation holds for the two patterns. An example would be that if two patterns have the same structure of their tactic representation, their relation is *is similar*. Table 3.3 shows all considered relations and the corresponding predicates for the patterns and their tactic representation. Every pattern is checked against all other patterns in the pattern system regarding all relations. The resulting patterns including their relations are shown in Figure 3.7.

Relation	**Description**	**Tactic Representation Predicate**
is an alternative	Patterns A and B solve the same problem, but propose different solutions.	$SourceNode(A) = SourceNode(B)$ **AND** $Graph(A) \neq Graph(B)$
uses	A sub-problem of pattern A is similar to the problem addressed by pattern B.	$Graph(A) \supset Graph(B)$
refines	Pattern B provides a more detailed solution than pattern A.	$SourceNode(A) = SourceNode(B)$ **AND** $Graph(A) \subset Graph(B)$
specializes	The solution of pattern B is a special case of the solution of pattern A. *Example: Pattern B specializes pattern A if they have the same graph structure, but pattern B uses a refined tactic where pattern A uses a more general tactic (e.g. B uses Replication Redundancy where A uses Redundancy).*	$Graph(A) \subset generalizedGraph(B)$
is similar	Patterns A and B provide the same solution to a similar problem *Example: Pattern B is similar to pattern A if they have the same graph structure and they use two related refined tactics. E.g. A uses Replication Redundancy and B uses Diverse Redundancy*	$generalizedGraph(A) \equiv generalizedGraph(B)$

Table 3.3: Description of pattern relations (slightly modified from [KP10b])

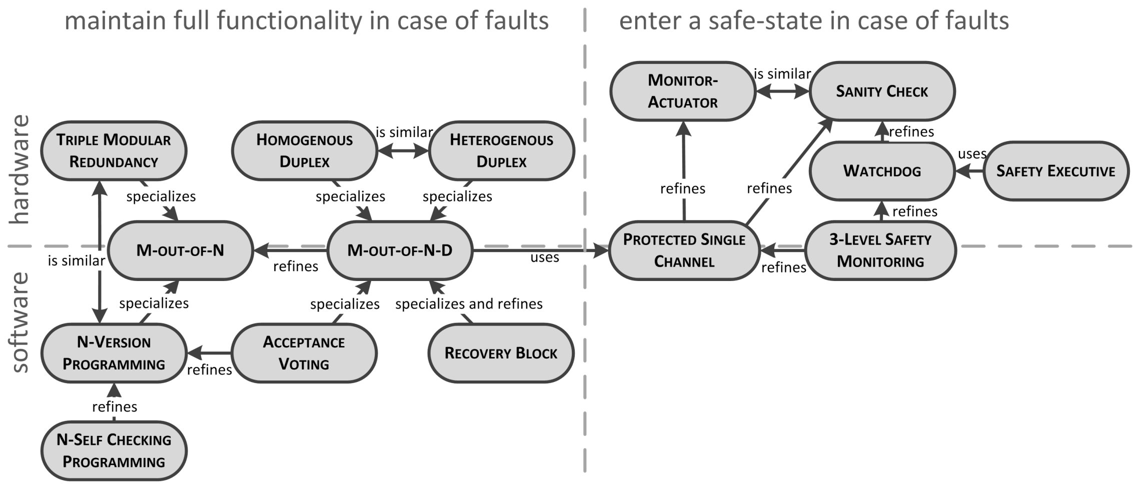
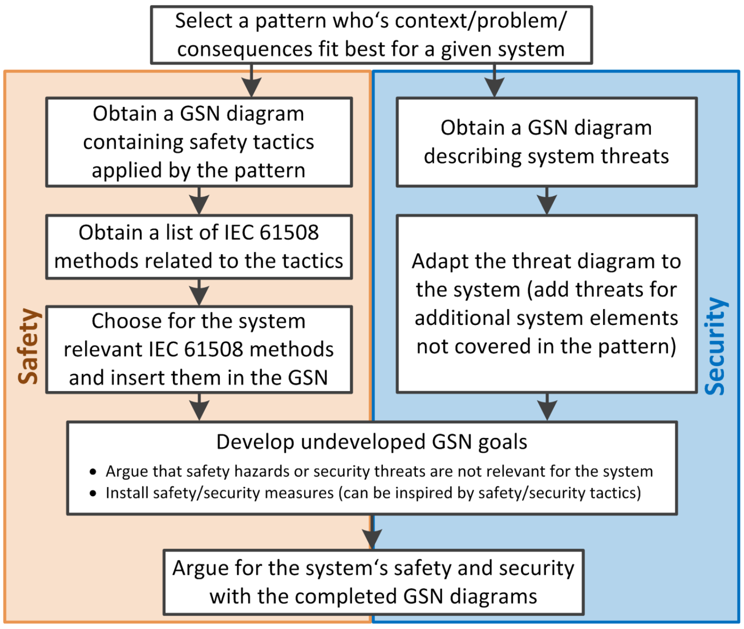


Figure 3.7: Overview of all patterns of the safety pattern system. The structurally identified relations are shown by arrows between the patterns.

Chapter 4

Application of the Patterns to a Case Study

This section first introduces an industrial case study and then applies the patterns to the case study and describes how the patterns help to construct a full safety and security argument in the form of GSN diagrams. Figure 4.1 gives on overview of steps for the pattern application which are described in detail in this section and in **Publication 4**.



Select a pattern who‘s context/problem/ consequences fit best for a given system
Safety
Security
Obtain a GSN diagram containing safety tactics applied by the pattern
Obtain a list of IEC 61508 methods related to the tactics
Choose for the system relevant IEC 61508 methods and insert them in the GSN
Obtain a GSN diagram describing system threats
Adapt the threat diagram to the system (add threats for additional system elements not covered in the pattern)
Develop undeveloped GSN goals
• Argue that safety hazards or security threats are not relevant for the system
• Install safety/security measures (can be inspired by safety/security tactics)
Argue for the system‘s safety and security with the completed GSN diagrams

Figure 4.1: Overview of steps described in this chapter to apply a safety pattern system with security aspects (based on **Publication 4**).

4.1 Case Study Description

The considered system is part of the HIPASE project which was already to some extent described in Section 1.2.1. The HIPASE project is conducted by the company Andritz Hydro and targets the development of a domain-specific automation system for hydro-power plants. This includes the design and development of the embedded device controlling the actual hardware in the power plant, the design and development of a software which communicates with this embedded device and which can configure the embedded device, and the design and development of software for higher-level control stations which can coordinate different hydro-power plants.

The part of the HIPASE project which is addressed in this thesis is the design of the embedded device that is installed in the power plants to control the hardware. This device will hitherto be referred to as the HIPASE-device. The HIPASE-device has to achieve the following four main functionalities:

- *Turbine Control* - The device has to sense the current speed of the turbine and has to control it in order to control the frequency of the generated energy. To control the turbine speed, the device controls magnetic valves to open and close the water supply to the turbine.
- *Protection* - The device has to sense voltages and currents in several components of the hydro-power plant and has to shut off systems if overvoltages or overcurrents are detected in order to not negate any mechanical damage to the system.
- *Synchronization* - When connecting the hydro-power plant to the energy grid, the generated power must have the same frequency and phase shift as the power grid. If the hydro-power plant is connected to the grid without adjusting these values, mechanical damage could result. Thus, the device has to monitor the frequency and phase values from the grid and only allow the hydro-power plant to connect to the grid if frequency and phase value differences are sufficiently small.
- *Excitation* - The device should maximize the produced energy, while still operating the power plant components in a way that ensures they do not get damaged. The device controls the current flowing through the generator which creates a magnetic field. Amongst others, this magnetic field strength determines the amount of energy produced. However, the field strength can only be increased to a value which does not damage the generator due to too high temperatures as a result of the flowing current. The device has to control the value of the magnetic field so that it is maximized, but the generator still receives no damage.

These four main functionalities are to some extent standardized. This means that there are to some extent sensor elements available which communicate via a standardized protocol particularly defined for the energy substation domain. This protocol is defined in the IEC 61850 standard and uses so-called *merging units* to collect sensor data and send them via Ethernet to the HIPASE-device. Apart from these standardized sensors, other sensors and actuators are directly connected to the HIPASE-device. Also, there is a network interface to the HIPASE-device, because an operator of the power plant should be able to interact with the device. Such interactions include parameterizing the control functionality of the HIPASE-device, shutting it off manually, or installing firmware updates. The above stated interfaces result in the minimum interfaces and environment for the HIPASE-device shown in Figure 4.2.

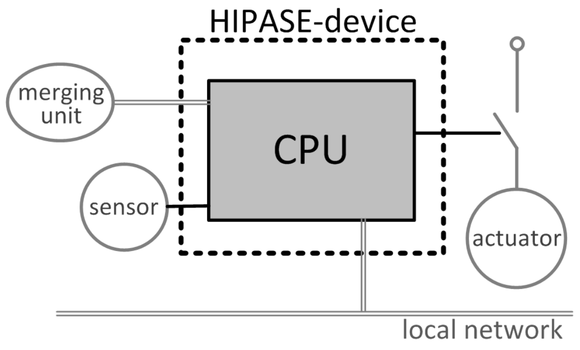


Figure 4.2: Basic system architecture and interfaces of the HIPASE-device not yet considering safety

An additional requirement for the HIPASE-device is that it has to be designed with respect to the IEC 61508 safety standard. The device has safety-critical functionality, because there are people operating the power plant who could get injured if the device malfunctions and because a malfunction could result in severe damage to the power plant hardware leading to considerable financial losses. Thus, an appropriate system architecture has to be found which improves the safety of the system and which complies with the IEC 61508 safety standard.

4.2 Pattern Selection

The main requirements for architecture selection come from the IEC 61508 safety standard. The HIPASE-device has to be developed according to IEC 61508 SIL3 which requires high fault coverage of single components and which practically requires some kind of hardware redundancy. Another requirement for the HIPASE-deivce is that it has to be resistant against faults up to some level and should thus also provide high availability. In the pattern map shown in Figure 3.7 on page 41 the patterns which are appropriate for these requirements are all patterns falling into the categories “hardware” and “maintain full functionality in case of faults”, which are:

- Homogenous Duplex Pattern. The pattern suggests having two identical redundant channels. The result of the redundant channels can be voted and a failure detection unit tells the voter if one of the channels appears to be erroneous to be excluded from the vote.

- Heterogenous Duplex Pattern. This pattern is the same as the Homogenous Duplex Pattern, but it uses diverse channels.

- Triple Modular Redundancy Pattern This pattern uses three channels and a majority voter device for the correct result.

- M-out-of-N Pattern. This pattern has N identical or diverse channels. A voter votes for the output provided at least by M of these N channels.
- M-out-of-N-D Pattern. This is the same as the M-out-of-N pattern, but additionally each channel can be diagnosed by a fault detector unit. The fault detector unit tells the voter whether certain channels have to be excluded from the vote.

The development and production cost for the HIPASE-device should be kept as low as possible. Therefore, just the Homogenous Duplex Pattern is selected, because it fulfills the safety requirements of redundant hardware, it does not require the development of redundant channels (no additional development cost), and it only requires two hardware channels. Figure 4.3 shows the core elements of the applied Homogenous Duplex Pattern.

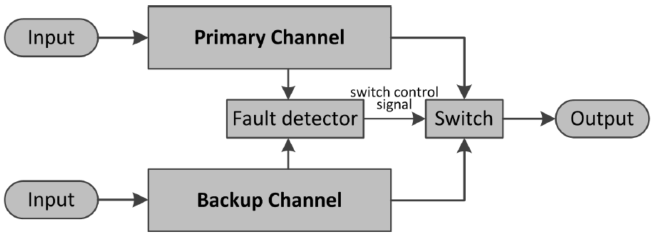
Pattern Name	Homogenous Duplex Pattern	**Pattern Type**	hardware, failover
Context	A safety-critical application without a fail-safe state has a high random error rate and a low systematic error rate.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components		
Forces	- the system cannot shut down because it has no safe state - development costs should not increase - safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime		
Solution	The system consists of a *Primary Channel* (active) and a *Secondary Channel* (backup) which are two identical hardware modules. A *Fault detector* monitors the channels and controls a *Switch* to select the *Backup Channel* in case of a *Primary Channel* failure.  Input Primary Channel switch control signal Fault detector Switch Output Input Backup Channel		
Consequences	Systematic and random faults in a single channel are detected and masked. System reliability strongly depends on the fault coverage of the fault detection unit and on the proper functionality of the switch.		

Figure 4.3: Main elements of the Homogenous Duplex Pattern. Instead of a switch, for this pattern, a voting element can also be used for whichever requires two correct inputs, or one correct input and one from the fault detector excluded channel.

4.3 Application of the Pattern

The Homogenous Duplex Pattern is applied to the HIPASE-device. The main component of the HIPASE-device which is error prone is the CPU. Therefore, two redundant CPUs are applied and as suggested by the pattern, both CPUs get their separate input values. The HIPASE-device also contains a fault detection unit which monitors the operation of these two CPUs. This fault detection unit is realized as a watchdog which periodically sends challenges to the two CPUs to find out whether they are still alive. Instead of the switch component, a voting functionality is realized. The outputs for the actuators are binary values and, depending on the type of actuator, the voting is realized with two relays arranged serial or parallel. This kind of voting means that for some actuators both CPUs have to actuate the output (serial) in order to operate the actuator, or just one CPU has to actuate the output (parallel) to operate the actuator. The fault detector components have the possibility to exclude CPUs from the voting procedure by simply cutting off their power supply or restarting them. Figure 4.4 shows the resulting HIPASE-device architecture.

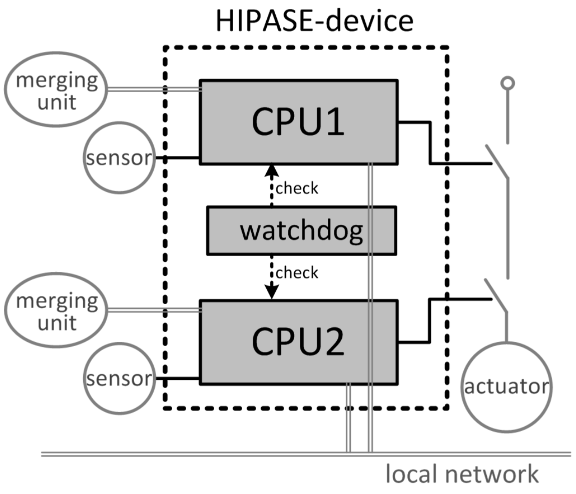


Figure 4.4: HIPASE-device architecture after applying the Homogenous Duplex Pattern. The switch component of the pattern has been replaced by the hardware voting functionality realized as relays.

4.4 Construction of a Safety Argument

4.4.1 GSN from the Pattern

The safety patterns each provide a GSN diagram to be used as a basic skeleton to construct a safety argument. Figure 4.5 shows this safety GSN skeleton for the Homogenous Duplex Pattern.

The GSN diagram shows how the main goal of the system, which is to maintain its safety functionality, is achieved. The GSN diagram shows the safety tactics in the form of GSN strategy elements. For the Homogenous Duplex Pattern, these three tactics are:

- *Replication Redundancy.* A redundant system is introduced with the aim of detecting or masking random hardware failures.
- *Override.* A failure of a subsystem is masked by overwriting its output with a more reliable or more safe value.

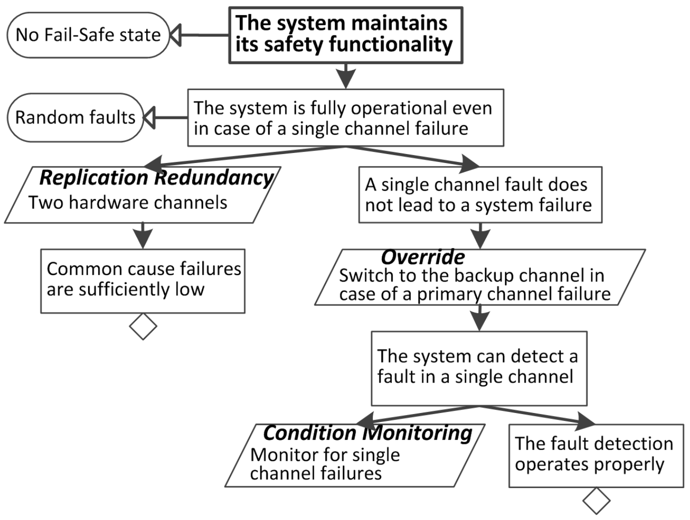

No Fail-Safe state
The system maintains its safety functionality
Random faults
The system is fully operational even in case of a single channel failure
Replication Redundancy
Two hardware channels
A single channel fault does not lead to a system failure
Common cause failures are sufficiently low
Override
Switch to the backup channel in case of a primary channel failure
The system can detect a fault in a single channel
Condition Monitoring
Monitor for single channel failures
The fault detection operates properly

Figure 4.5: Safety GSN diagram of the Homogenous Duplex Pattern. The GSN contains the tactics applied by the pattern: *Replication Redundancy*, *Override*, and *Condition Monitoring*

- *Condition Monitoring.* Deviations of intended system outputs or system states are monitored and detected.

4.4.2 Choosing appropriate IEC 61508 Methods related to the Tactics

The safety tactics were linked to specific methods suggested to be applied by the IEC 61508 standard in **Publication 1**. The relevant IEC 61508 methods related to the tactics *Replication Redundancy*, *Override*, and *Condition Monitoring* are shown in Table 4.1. This set of IEC 61508 methods serves as a base to select appropriate methods for the HIPASE-device architecture. The IEC 61508 safety standard overall provides more than 200 such methods to choose from. Thus this tactic-based pre-selection provides a good basis to chose the first and most important methods. The methods actually applied for the HIPASE-device architecture were discussed and selected during a set of expert meetings by the company Andritz Hydro and the approach described in this thesis does not provide further guidance on this step. The methods which were selected in the first meeting and which were considered as most important are marked with a checkmark in Table 4.1. These methods are then used to complete the safety GSN skeleton of the pattern.

4.4.3 Elaborating the Safety GSN Diagram

The next step in obtaining a GSN diagram to allow safety reasoning for the HIPASE-device architecture is to include the selected IEC 61508 methods into the safety GSN skeleton. For that, the GSN strategies describing the tactics are simply replaced by GSN strategies describing the IEC 61508 methods. Figure 4.6 shows the completed GSN diagram for the HIPASE-device architecture where additionally added elements are shown in orange and with dashed lines. In addition to including the IEC 61508 methods, the GSN diagram has to be elaborated so that evidence for the achievement of the goals is included. Such evidence could be a simple argumentation, could be test results, or could be some kind

Tactics	**Related IEC 61508 methods**	
Replication Redundancy	A.2.1 Test by redundant hardware	✓
	A.2.5 Monitored redundancy	
	A.3.5 Reciprocal comparison by software	
	A.4.5 Block replication	
	A.6.3 Multi-channel output	
	A.6.5 Input comparison/Voting	✓
	A.7.3 Complete hardware redundancy	✓
	A.7.5 Transmission redundancy	
Override	A.1.3 Comparator	
	A.1.5 Idle current principle	✓
	A.8.1 Overvoltage protection with safety shut-off	
	A.8.3 Power-down with safety shut-off	
Condition Monitoring	A.1.1 Failure detection by online monitoring	✓
	A.6.4 Monitored output	
	A.8.2 Voltage control	
	A.9.1 Watch dog with separate time base without time-window	
	A.9.2 Watch dog with separate time base and time-window	✓
	A.9.3 Logical monitoring of program sequence	✓
	A.9.4 Temporal and logical program sequence monitoring	✓
	A.9.5 Temporal monitoring with on-line check	✓
	A.12.1 Reference sensor	
	A.13.1 Monitoring	

Table 4.1: Tactics and their related IEC 61508 methods. The column on the right shows a checkmark for all methods that are selected to be applied by the HIPASE-device architecture (Table partially based on **Publication 1**)

of documentation. For example, in Figure 4.6, the *"The fault detection operates properly"* goal is ensured by the fact that the watchdog used for the HIPASE-device applied internal self-tests and these self-tests lead to a high fault coverage of the watchdog itself. The evidence for that has to be documented (e.g. fault coverage tests for the watchdog) and a link to this evidence or the argument that there is such evidence is included in the GSN diagram.

The method for finding this evidence is not part of the method described in this thesis and for the HIPASE-device architecture, the way of providing evidence was discussed in expert meetings and then documented later on in the GSN diagram. Now, this complete GSN diagram relates the actually implemented IEC 61508 methods to the overall goal of maintaining the safety functionality. Thus, this diagram can be used to argue why the methods are used and how they achieve the overall system goal.

Of course this diagram just argues for safety of the high-level architecture and the elements of the HIPASE-device (like for example the CPUs) contain further safety measures. These elements could again be refined by applying a safety pattern. For example, the software running on the CPU could be constructed using a safety pattern to then construct a safety GSN for the software also. However, in this example, just the application of the patterns to the high-level architecture is shown.

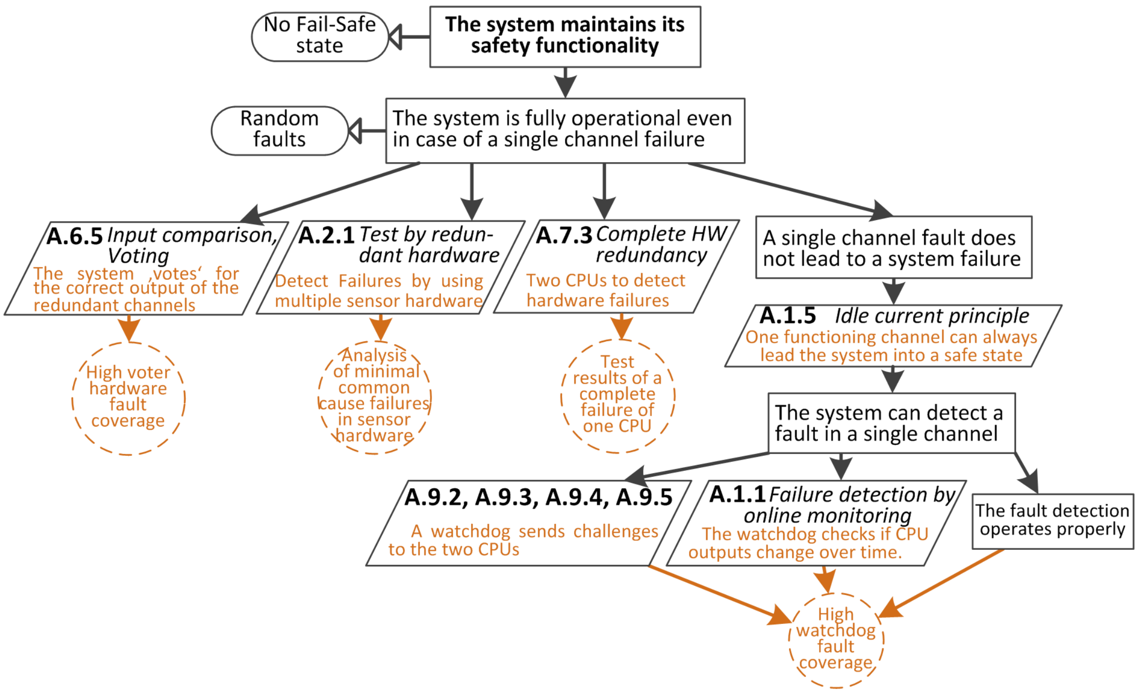

No Fail-Safe state
The system maintains its safety functionality
Random faults
The system is fully operational even in case of a single channel failure
A.6.5 Input comparison, Voting
The system ‚votes' for the correct output of the redundant channels
A.2.1 Test by redundant hardware
Detect Failures by using multiple sensor hardware
A.7.3 Complete HW redundancy
Two CPUs to detect hardware failures
A single channel fault does not lead to a system failure
A.1.5 Idle current principle
One functioning channel can always lead the system into a safe state
High voter hardware fault coverage
Analysis of minimal common cause failures in sensor hardware
Test results of a complete failure of one CPU
The system can detect a fault in a single channel
A.9.2, A.9.3, A.9.4, A.9.5
A watchdog sends challenges to the two CPUs
A.1.1 Failure detection by online monitoring
The watchdog checks if CPU outputs change over time.
The fault detection operates properly
High watchdog fault coverage

Figure 4.6: Elaborated Safety GSN diagram for the HIPASE-device architecture. All the tactics are replaced by actual IEC 61508 methods and all goals are connected to evidence elements.

4.5 Construction of a Security GSN Diagram

4.5.1 GSN from the Pattern

The patterns each provide a GSN diagram representing all security threats which could bring the system into a safety-critical state. Figure 4.7 shows the security GSN diagram of the Homogenous Duplex Pattern. The GSN diagram has the main goal of maintaining the safety of a system even in case of attacks. The subgoals address the prevention of exploits related to security threats. All the subgoals are modeled as undeveloped GSN goals. This means that all these subgoals have to be developed in order to obtain a complete argument for the security of the system.

4.5.2 Elaborating the Security GSN Diagram

To elaborate the security GSN diagram, all threats of the diagram have to be considered and either an argument has to be found and documented as to why the threat is irrelevant for the system or the threat has to be mitigated. Figure 4.8 shows the completed security GSN diagram with all added elements marked blue and with dashed lines.

Switch component. The threats to the *Switch* component are not relevant for the HIPASE-device architecture, because instead of a switch component, there is actual hardware in the form of relays and the actuators are directly wired to these relays. Thus, under the assumption that an attacker has no physical access to the hardware, the attacker cannot spoof the relay component, cannot mount a denial

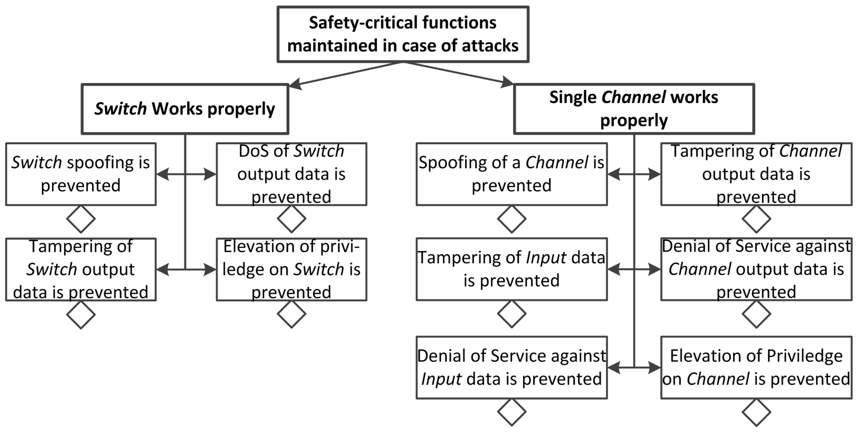
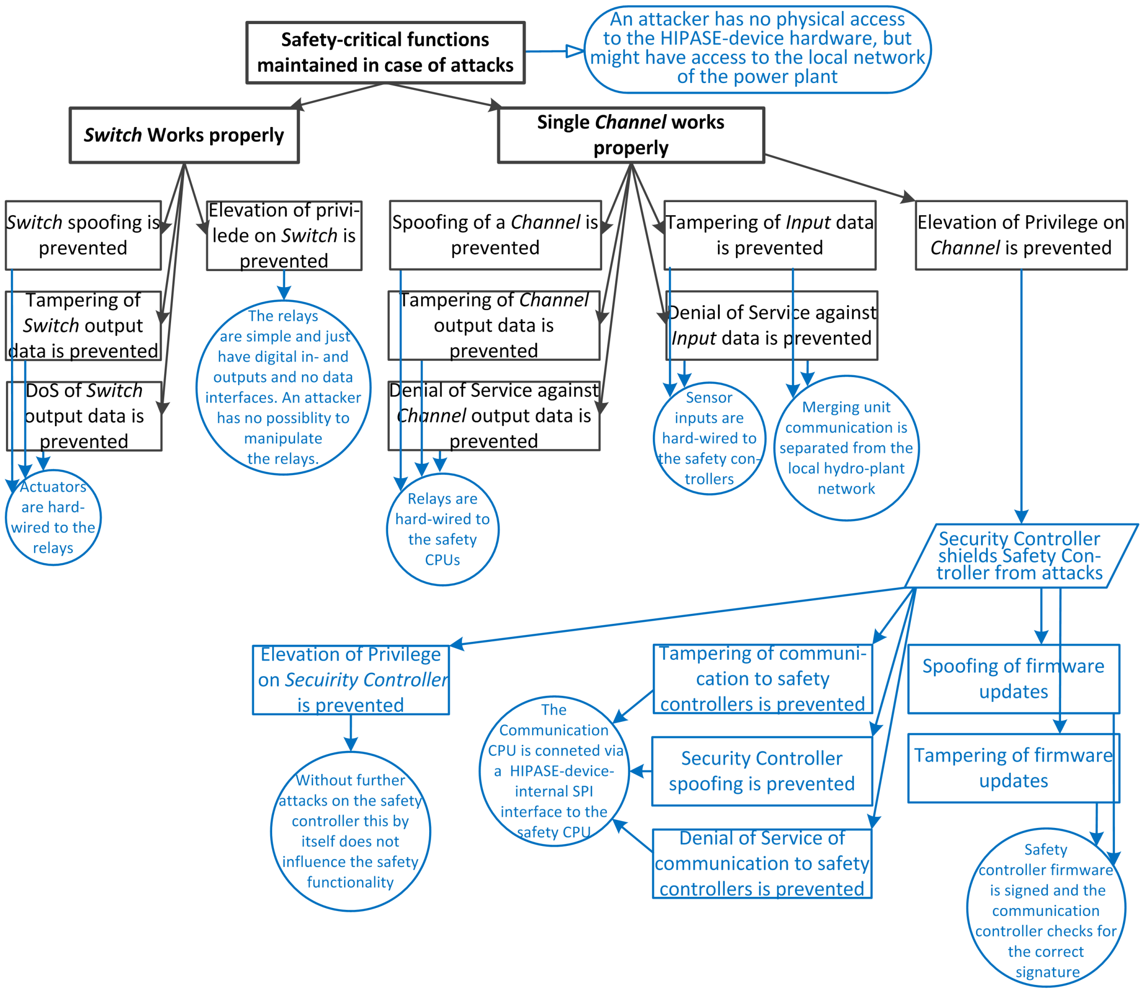

Safety-critical functions maintained in case of attacks
Switch Works properly
Single *Channel* works properly
Switch spoofing is prevented
DoS of *Switch* output data is prevented
Tampering of *Switch* output data is prevented
Elevation of privi-ledge on *Switch* is prevented
Spoofing of a *Channel* is prevented
Tampering of *Channel* output data is prevented
Tampering of *Input* data is prevented
Denial of Service against *Channel* output data is prevented
Denial of Service against *Input* data is prevented
Elevation of Priviledge on *Channel* is prevented

Figure 4.7: Security GSN skeleton of the Homogenous Duplex Pattern. The GSN diagram contains security threats which could influence the safety functionality.

of service attack on the relay output, and cannot tamper with the output value. Elevation of privilege on the system is also not possible, because the relays form a very simple construct. Thus, because of this simple design, it can be assumed that an attacker cannot elevate his privileges to modify the systems behavior.

Single Channel. Most of the threats for a *Single Channel* are not relevant for the HIPASE-device. The denial of service and tampering threats related to channel output data as well as the spoofing threat are not relevant, because the channel outputs are hardwired to the relays. However, the threats related to the channel input data are relevant. Input data comes from directly wired sensors and from the merging unit. Under the assumption that an attacker has no physical access to the hardware, the threats related to directly wired sensors are not relevant. However, the sensor data obtained from a merging unit is sent via Ethernet. If an attacker can manipulate or block the Ethernet traffic, the sensor data transmission can be influenced. Thus, for the HIPASE-device architecture, the merging unit is connected via a separate Ethernet to the HIPASE-device. This Ethernet connection is not connected to any other network elements in the hydro power-plant and thus an attacker cannot tamper with the data sent from the merging unit or with the merging unit itself.

Elevation of Privilege on Single Channel. The most severe threat for the HIPASE-device is *Elevation of Privilege on a Channel.* The HIPASE-device CPUs are directly connected to the local power plant network, because firmware updates have to be possible and because the HIPASE-device has to provide status information from the power plant to the power plant operator who uses a computer in the local power plant network to visualize the power plant status. The measure taken for the HIPASE-device is to reduce the attack surface by including an additional CPU (hitherto called communication-CPU) which handles all external communication to the local power plant network. This communication-CPU provides an additional line of defense meaning that an attacker first has to successfully mount an attack on this CPU in order to go on attacking the actual safety-relevant CPUs of the HIPASE-device. The only data which can still directly affect the safety CPUs of the HIPASE device are the firmware updates. In the case of a tampered firmware update for the safety CPUs, an attacker could elevate his privileges on this CPU. To mitigate this threat, the HIPASE-device only allows signed firmware updates in order to guarantee the authenticity of installed updates. The communication CPU checks the signature of the firmware updates and only sends them to the safety CPUs, if the signature check succeeds.



Safety-critical functions maintained in case of attacks
An attacker has no physical access to the HIPASE-device hardware, but might have access to the local network of the power plant
Switch Works properly
Single *Channel* works properly
Switch spoofing is prevented
Elevation of privilede on *Switch* is prevented
Spoofing of a *Channel* is prevented
Tampering of *Input* data is prevented
Elevation of Privilege on *Channel* is prevented
Tampering of *Switch* output data is prevented
The relays are simple and just have digital in- and outputs and no data interfaces. An attacker has no possiblity to manipulate the relays.
Tampering of *Channel* output data is prevented
Denial of Service against *Input* data is prevented
DoS of *Switch* output data is prevented
Denial of Service against *Channel* output data is prevented
Sensor inputs are hard-wired to the safety controllers
Merging unit communication is separated from the local hydro-plant network
Actuators are hard-wired to the relays
Relays are hard-wired to the safety CPUs
Security Controller shields Safety Controller from attacks
Elevation of Privilege on *Secuirity Controller* is prevented
Tampering of communication to safety controllers is prevented
Spoofing of firmware updates
The Communication CPU is conneted via a HIPASE-device-internal SPI interface to the safety CPU
Security Controller spoofing is prevented
Tampering of firmware updates
Without further attacks on the safety controller this by itself does not influence the safety functionality
Denial of Service of communication to safety controllers is prevented
Safety controller firmware is signed and the communication controller checks for the correct signature

Figure 4.8: Completed Security GSN argument for the HIPASE-device. No more GSN goals are undeveloped and all goals are linked to evidence elements.

4.5.3 Influence on the Safety Functionality

For security reasons, an additional element was added to the HIPASE-device architecture. A communication CPU was placed between the local power plant network and the safety

CPUs. This communication CPU could have an effect on the safety functionality of the system and if that is the case, the communication CPU also has to be considered in the safety GSN argument.

The IEC 61508 safety standard requires any interaction between a system which is connected to a safety-related system to be analyzed in order to find out whether the safety functionality really does not depend on the other system. For the HIPASE-device this means that the safety CPUs have to be able to operate without the communication CPU and that the communication CPU cannot bring the safety CPUs into a critical state. Therefore, the HIPASE-device is designed in a way that the communication CPU only indicates upon request if a new firmware for the safety CPUs is available and the safety CPUs decide themselves if they want to apply the update. This means that the communication CPU cannot trigger the update. This is important, because otherwise, the communication CPU could interrupt the safety CPU during important safety-related operations. The firmware updates sent from the communication CPU to the safety CPUs are protected with a CRC field so that the data integrity can be checked locally by the safety CPUs. Other communication between the power plant network and the safety CPUs is related to the visualization data sent from the safety CPUs to the network. Thus, for this communication the communication CPU has to make sure that data is just sent form the safety CPU to the network and not in the other direction.

By these simple measures, the safety CPUs are made independent from the communication CPU and thus the communication CPU does not have to be included in the safety GSN argument. For more complicated cases, advice on more finding out whether a safety component is independent from another or not can be found in **Publication 9**.

Chapter 5

Evaluation of the Patterns and the Pattern-Based Development Method

This section evaluates the proposed security enhanced safety patterns and their method of application. First, the benefits and liabilities of applying the method and the patterns to the HIPASE-device are discussed based on feedback from Andritz Hydro. Next, the method is compared to other pattern-based safety development methods to show in which aspects the proposed method and patterns are better compared to existing methods. Finally, the safety and security aspects of the patterns are quantified and measured.

5.1 Discussion and Benefits for the HIPASE Project

The proposed method of applying safety patterns to find an appropriate safety architecture and of constructing GSN diagrams for safety and security was applied for the HIPASE-device and the developers were informally questioned regarding the benefits and shortcomings of the method. Some of the feedback has already been used to enhance the method to the current version as it is now. The following main points were gathered from the feedback:

Safety Patterns. All architects agreed that it is important and interesting to have a collection of possible safety architectures. However, the senior architects who are safety experts, did not gain fundamental additional knowledge from the problem/solution section of the patterns. Still, for more junior architects the patterns provided a valuable overview of safety architectures and the textual pattern sections such as the forces or consequences brought additional insights.

Pattern Selection. Finding an appropriate pattern for the HIPASE-device was quickly done. The categories hardware/software and failover/fail-safe helped to quickly reduce the patterns to a suitably small set.

GSN. The Goal Structuring Notation was new to the system architects at Andritz Hydro and they were interested in learning a new way to structurally present their safety measures in a system to the safety certification organization. The system architects quickly understood the main concepts of GSN.

Safety GSN Diagram. The safety GSNs contain safety tactics which are linked to IEC 61508 methods. This link was considered by some developers as useful, while others considered them as not necessary. More specifically, for employees which were new to the topic of safety, the IEC 61508 method suggestions by the patterns were considered as beneficial. For safety reasoning, the safety GSN based on the safety skeleton of the pattern was first accepted by Andritz Hydro and considered as very useful, because up to that time there was no other structured method used to present safety evidence for the whole system. However, during the development of the project, the safety GSN was not updated by Andritz Hydro and ran out of sync with the project. From that point on, the safety GSN was still used to argue why some of the IEC 61508 methods were chosen to realize the architecture, but it was not used anymore to present a complete safety argument for the HIPASE-device. Instead, a list of textual safety requirements was maintained, because it was less effort. The disadvantage of that is that it is not easy to see what the purpose of some of the safety requirements are and how they are related. However, for the HIPASE project, that was not considered as important, because the senior safety architects of the project had already constructed several similar systems and thus could base the HIPASE requirements on requirements from previous projects and they could also use existing (textual) safety arguments from previous projects.

Security GSN Diagram. At first, Andritz Hydro did not plan to extensively consider security aspects for the HIPASE-device. However, that changed during the beginning of the architecture design. The pattern-based safety development method brought their attention to security issues. It was a great benefit to have the security GSN as a starting point to see which security threats are relevant for the safety functionality of the system. A big advantage was that the security threats were conveyed in a way that was at that point already familiar to the system architects: in GSN. This was especially important, because the architects were not security experts and thus it would at that point have been an additional problem to introduce new security methods or notations (as it would be the case for regular STRIDE analysis including threat trees). At Andritz Hydro, security considerations increased steadily during the development of the HIPASE-device. Currently, the company maintains lists of security threats based on the threats of the security GSN skeleton of the pattern. Andritz Hydro broke the specific threats further down to a level of security requirements and thus have a complete security argument about how specific security requirements fulfill the overall aim that attackers cannot influence the safety functionality of the system.

5.2 Comparison to other Pattern-Based Safety Development Methods

This section compares the proposed method of applying the safety patterns and constructing GSN diagrams with the help of these patterns to other pattern-based safety development methods from literature. Such a comparison regarding safety aspects is already described in **Publication 5**. In addition, this section also compares security aspects of the pattern-based safety development methods.

For the comparison the following pattern-based safety development methods, which are described in more detail in Section 2.2.3 and **Publication 5**, are considered:

- Applying Patterns along the IEC 61508 Safety Lifecycle [RVK13b]
- Safe Control System Method [HS13]
- TERESA Project [HDGJ10]
- SIRSEC Project [RHFP13]
- Safety Tactic-Based Approach [Wu07]
- Safety Architecture Patterns [Arm10]
- Safety Architecture Patterns + UML [AKA12]
- SDL Design Patterns [FGG$^+$05]
- REFLECT [PKCD11]
- AltaRica Safety Patterns [KSB$^+$04]
- Safety Timing Templates [Bit07]

The methods are compared regarding general pattern-related criteria like, for example, the type of patterns they use. Other pattern-related criteria address the application of the patterns like for example, the support of the method for pattern search, selection, and integration or the tool support of the method in general. Regarding safety, the methods are compared regarding the safety domain they address and their conformance to safety standards, and regarding their support for safety reasoning. Security aspects and the support regarding security issues is discussed for the different methods and the maturity of the method is assessed in terms of their applicability to larger systems or existing evaluations of the methods. Table 5.1 shows an overview of the comparison. More detailed discussions about the presented pattern-based development methods and about the content of the table is provided in **Publication 5**.

	Pattern	Pattern Application			Safety-Process Aspects		Security	Method Maturity	
	Type	**Pattern-Search/Sel.**	**Pattern Integration**	**Tool Support**	**Standards, Dev.-Process**	**Safety Reasoning / Traceability**	**Security**	**Applied to**	**Evaluation**
Patterns - Safety Lifecycle	process and architecture patterns	development-phase dependent pattern links	-	-	development process based on IEC 61508	some patterns cover verification activities	-	-	-
Safe Control System Method	safety process and safety architecture patterns	shows applicable safety patterns along the development process	UML component and sequence diagrams with defined interfaces for some patterns	-	-	GSN patterns; pattern interfaces provide link to requirements	-	made-up system	informal general discussion
TERESA project	model-based security and dependability patterns	text- or category-based pattern search tool	common pattern meta-model with defined interfaces; pattern instantiation with help of tools	pattern search/ selection/repository tool; pattern editor; property and constraint editor; process editor	meta-model for safety-process; models present for railway-safety and IEC 61508	the process meta-model supports checkpoints to define requirement check activities	the main focus is on security patterns, but not combined with safety	industrial system	discussion; risk estimations with/without TERESA
SIRSEC Project	safety architecture patterns	developer chooses from pattern categories	patterns have defined interfaces; integration with tool support	Eclipse/Papyrus tool for pattern selection and instantiation	-	patterns contain links to requirements	-	made-up system	-
Safety Tactic-Based Approach	safety tactics	tactics provided to counter elements of negative scenarios	-	-	tactic application along the safety-process discussed	GSN patterns used to build safety cases	-	industrial system	discussion on application to case studies
Safety Architecture Patterns	safety architecture patterns	tool asks questions to find an appropriate pattern	-	tool to guide pattern selection and to calculate reliability and risk reduction	pattern recommendation for appropriate IEC 61508 SIL	-	-	small proof-of-concept system	-
Safety Architecture Patterns + UML	safety architecture patterns	-	patterns with defined UML interfaces are integrated in existing architecture model	Enterprise Architect extension to manage, view, and retrieve patterns (UML)	-	-	-	-	-
SDL Design Patterns	SDL patterns	-	interfaces and descriptions of how to adapt SDL diagrams	-	-	verification of timing behavior	-	industrial system	-
REFLECT	safety architecture patterns	automatic pattern selection according to requirements	code transformation based on annotated input source	code transformation tool using patterns to fulfill safety requirements	-	requirements linked to implementation	-	made-up system	-
AltaRica Safety Patterns	safety architecture patterns	-	mapping of pattern input/output interfaces	architecture modeling and model checking tools	-	requirements linked to implementation	-	industrial system	discussion on application to case studies
Safety Timing Templates	temporal safety templates	question/answer-based guidance	mapping of pattern input/output interfaces	architecture modeling and model checking tools	-	requirements linked to implementation	-	industrial system	discussion on application to case studies
Method described in this Thesis	safety architecture patterns and safety tactics	graphical representation of pattern relationships	-	-	patterns linked to IEC 61508 methods	GSN diagrams for patterns	threat analysis of safety patterns and GSN diagram for security reasoning	industrial system	metrics

Table 5.1: Comparison of pattern-based safety methods with focus on their safety and security aspects (based on **Publication 5**)

Most of the methods are either related to a safety standard, or provide some general support for safety such as safety reasoning by constructing safety cases or traceability support to link safety implementations to safety requirements. It is not surprising that such things are part of the methods, because all of the methods are actually intended to be used for safety-critical system development.

The main benefits and differences of the proposed pattern-based safety development method compared to the other methods covered in literature are below:

Security. There are just two methods in Table 5.1 which cover security aspects. These are the method described in this thesis and the TERESA approach. The TERESA approach actually mainly focuses on security and only also addresses safety issues to some extent. Most of the provided patterns by TERESA actually address security problems and only a few of them address safety issues. The TERESA approach provides tools and a concept for designing, managing, integrating, and reasoning about these patterns independent from whether they address security or safety. However, TERESA does not cover security and safety issues together for the presented patterns. This is the main benefit of the method and the patterns presented in this thesis, because the method from this thesis takes safety patterns and considers security aspects for these patterns. Thus, from the pattern-based safety development methods shown in Table 5.1, this method is the only one which introduces security aspects into pattern-based safety development.

Collection of Safety Patterns. Many of the methods from Table 5.1 describe the approach of how to apply safety patterns and how to reason about safety. However, most of the methods just describe a few actual safety patterns. Several methods do provide links to a larger set of patterns which could be integrated into the tools or pattern form in order to be used by the method; however, that requires additional effort and might not always be a straight forward task. The method described in this thesis mainly focuses on redundancy-based architecture patterns and fully describes 15 such patterns including their GSN diagrams which help to reason for safety and security. These 15 patterns are collected from literature on patterns and pattern collections of high-level safety architectures.

Evaluation. It is difficult to evaluate development methods. It would be ideal to have two identical systems and let these systems be developed once with and once without the development method. However, such an evaluation is quite complicated and it is difficult to develop identical systems twice in a similar environment. This might be the reason that just few of the methods in Table 5.1 actually provide an evaluation. Some of them provide no evaluation at all and some discuss the application of the method to an example system and use the feedback on that application as an evaluation. The TERESA approach additionally provides a risk estimation for a project with and without applying the TERESA approach. This thesis uses for evaluation the feedback gathered from the application of the method to an industrial system, and the method additionally uses metrics related to safety and security (see the following sections) to quantitatively evaluate the patterns. Furthermore this comparison to other pattern-based safety development methods is used to show the benefits of the method and of the patterns described in this thesis.

5.3 Evaluation of the Safety Patterns

The main ideas of the safety patterns have been documented in literature for quite some time. Several of the patterns covered have already been mentioned in pattern form in 1998 [Dou98] and have been modified and re-published several times. Reports such as [Ham12] show that these patterns are actually applied in industry and that the main ideas of these pattern do provide benefits.

In this section, the additionally introduced safety-related parts of the patterns will be discussed:

- The Safety GSN Diagrams
- The Link to IEC 61508 Methods

5.3.1 The Safety GSN Diagrams

The safety GSN diagrams from the patterns are structurally developed from the pattern descriptions in literature. The GSN diagrams contain the safety tactics applied by the patterns and they contain scenarios which have to be fulfilled after applying the pattern.

Solely based on the tactics found for the patterns, the relations between these patterns were mined (see Figure 3.7 on page 41). The connections between the patterns appear to be reasonable. For example, according to the figure, the Homogenous Duplex Pattern is similar to the Heterogenous Duplex Pattern and both are specializations of the M-out-of-N-D Pattern. This appears quite sound, because the duplex patterns are actually some kind of 1-out-of-2 system with failure diagnosis. Also for the other patterns, the relations appear natural and sound. Thus, the selection of tactics included in the safety GSN diagrams of the patterns seems to be appropriate.

Apart from the safety tactics, the GSN diagrams contain safety-related scenarios relevant for the pattern. These scenarios were structurally mined from the pattern descriptions as suggested in [Bab07]. However, the fact that the scenarios were gathered with a structured method does not mean that these scenarios are complete. As it is always the case with patterns, this part is work in progress and during future development using the patterns, the patterns and their relevant scenarios might have to be updated. However, the general approach to build the GSN diagrams by considering all relevant scenarios as subgoals is an approach commonly applied for safety-critical systems [Wu07] and thus it appears that reasonably complete safety GSN diagrams were developed for the patterns.

5.3.2 Link to the IEC 61508 Standard

The tactics contained in the GSN diagrams contain links to methods suggested to be applied by the IEC 61508 safety standard. The standard provides more than 200 methods such as, for example, *"Monitored Redundancy"* or *"Code Protection"*. The proposed method in this thesis helps to select appropriate methods from this pool in order to implement a safety architecture based on a pattern. To evaluate whether the method suggestion based on the tactics is appropriate, *Precision* and *Recall* metrics are calculated for industrial projects applying one of the safety architectures described by the patterns. The metrics then indicate whether the suggestion of the methods from the patterns would have been complete for that architecture and how many other methods not suggested by the pattern were additionally required. The metrics are calculated with the following equations:

$$Precision = \frac{\text{Number of methods suggested by the pattern and applied}}{\text{Number of methods suggested by the pattern}} \tag{5.1}$$

$$Recall = \frac{\text{Number of methods suggested by the pattern and applied}}{\text{Number of methods applied by the architecture}} \tag{5.2}$$

As industrial projects, in addition to the HIPASE-device, one other system has been found in literature which provides detailed information about the applied IEC 61508 methods. The system describes the architecture of a frequency converter and also applies the Homogenous Duplex architecture. Details about the actually applied IEC 61508 methods of that architecture can be found in **Publication 6**. Table 5.2 shows the resulting precision and recall values for the frequency converter and for the HIPASE-device.

	Precision	**Recall**
HIPASE-device	$\frac{17}{22} = 77.2\%$	$\frac{17}{23} = 73.9\%$
Frequencey converter system	$\frac{15}{22} = 68.2\%$	$\frac{17}{23} = 68.2\%$

Table 5.2: Precision and recall values for the IEC 61508 method suggestion of the Homogenous Duplex Pattern

The resulting precision and recall values for the method suggestion are around 70%. This means that about two thirds of the methods suggested by the patterns are actually applied and about one third of the applied methods has to be additonally searched for in the IEC 61508 standard. This shows that the methods suggested by the patterns are by no means sufficient to construct a safety architecture; however, it also shows that quite a lot of the methods suggested are actually implemented. In particular, for people who are new to the safety domain and to the safety standard, this brings benefits in terms of good advice for which methods from the IEC 61508 standard to look at first.

5.4 Evaluation of the Security Aspects

The safety patterns contain a security GSN diagram representing security threats relevant to the safety functionality of the system. The security threats were analyzed using the STRIDE threat analysis approach. This approach is most commonly used for finding high-level security threats in a system. Thus, it can be argued that the considered threats for the patterns are likely to be complete, because a well-known approach is structurally applied to find relevant threats.

To evaluate the security improvement when applying one of the patterns, security risk estimation can be done. There has been a lot of work in literature on quantifying security aspects of a system and on calculating security metrics for systems [MFMP10]. Most of these quantitative methods are based somewhere on the estimation of the likelihood and impact of security attacks. This section outlines how such a security risk estimation can be applied to the safety patterns in order to evaluate the improvement or the impairment of using one of the patterns from a security point of view.

The security GSN diagram of the patterns consists of subgoals which represent threats. If the risk values for these threats are known, then the risk values can be propagated up in the GSN diagram and the overall security risk can be calculated. The GSN diagram itself actually already contains part of the risk information. The GSN diagram contains the information about which threats are directly safety-critical if exploited and which threats have to be combined in order to be safety-critical. Thus, in order to obtain a security risk value, simply the likelihood of an exploit has to be evaluated and not also the impact. When these likelihoods are propagated up the GSN diagram, one gets one single value representing the security risk for a system after applying the safety pattern.

Publication 8 shows the detailed security estimation for the initial HIPASE-device architecture before and after the theoretical application of safety patterns. Table 5.3 shows the resulting security risk improvement values of applying the safety patterns. The values are specific for the HIPASE-device and depend on the likelihood estimation for the security threats which are described in **Publication 8**. The absolute values in the table do not provide much detailed insight for the security of the system; however, it is interesting to see that the application of some of the patterns actually leads to a decrease in security. That is, because for these systems, additional redundant hardware is introduced and that increases the attack surface of the overall system. The values in Table 5.3 were used to argue for the application of a Heterogenous Duplex system for the HIPASE-device, because of its benefits for security. However, the Homogenous Duplex system was chosen by Andritz Hydro because of development cost reasons. Still, the security risk numbers can be used to indicate whether a safety architecture is good or bad for security and in case of the HIPASE-architecture the numbers contributed to the decision to have a separate communication CPU in order to shield the safety CPUs by reducing the attack surface of the overall system and by introducing an additional line of defense.

	Security Risk Improvement
Homogenous Duplex	0.99
Heterogenous Duplex	1.11
Triple Modular Redundancy	0.99

Table 5.3: Security risk improvement values for the HIPASE-device architecture when applying three initially considered safety patterns

The quantified security values evaluate the benefits of the safety patterns from a security point of view and they can be used to more easily argue about security considerations when deciding on a safety architecture.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presented a collection of safety patterns and showed with an industrial example system, how they can be applied. The safety patterns were partly collected from literature and partly mined from the IEC 61508 safety standard. The patterns were analyzed with a structured threat analysis from a security point of view. To enable security as well as safety reasoning, the patterns were equipped with a Goal Structuring Notation diagram.

From a patterns' point of view, the thesis provides a structured collection of safety patterns which is not yet present in literature. The patterns from literature were extended with regards to the following elements:

- *Quality attribute consequences.* The safety patterns from literature did not contain explicitly stated effects of the pattern application on quality attributes such as, for example, cost or performance. The pattern consequences were extended regarding these effects.
- *Security aspects.* The patterns describe the relevant security threats which could compromise the safety functionality of the system.
- *GSN diagrams.* The security aspects as well as the safety aspects of the patterns are organized in GSN diagrams to allow reasoning and the structured presentation of arguments as to why an implementation of the pattern is secure or safe.
- *Known uses.* The patterns were extended to include three brief descriptions with references to applications of the pattern.
- *Pattern relations.* A structured method for finding the relationships between the safety patterns is applied. The method has yet only been applied in literature to Gang of Four design patterns. Thus, the presented safety pattern system shows the broader applicability of this method.

From a safety point of view, the thesis provides a method to guide the development of safety systems and to link the high-level design of the system to specific methods proposed by the IEC 61508 safety standard. This link is present in Goal Structuring Notation diagrams which contain safety tactics linked to the IEC 61508 standard. The link provides the following benefits:

- *System development.* The development of a system can be guided by the patterns and the link to IEC 61508 methods provides further information on how to implement a specific pattern, because the IEC 61508 methods are described in the IEC 61508 standard which also provides references to further information regarding the implementation of the methods. The application of methods suggested by the safety standard provides advantages during safety certification, because the certification authorities have already approved of these methods.

- *Safety reasoning.* The link from the patterns to the IEC 61508 methods can be used for safety reasoning. The link shows how the high-level aim of meeting the safety requirements of a system is achieved via specific methods and measures suggested by the safety standard. Thus, the link can be used during safety certification to argue why a selected method has been chosen.

From a security point of view, the thesis brings the big advantage that the patterns introduce security to the safety domain. Security is not yet thoroughly integrated into safety engineering and is a rather new field for companies who in the past have just worked on safety-critical systems which were not exposed to many security threats, because, for example, these systems might not have been connected to the Internet. However, this field changes as more and more devices become interconnected and thus the need for the consideration of security for such systems rises. The patterns provide a connection between safety best practice in the form of architectural safety solutions and security best practice in the form of threat analysis and elaboration of countermeasures for these threats. Thus the patterns help on the one hand to point out that there are security threats for safety critical systems and on the other hand to provide guidance on how to find and apply countermeasures to mitigate these threats.

6.2 Directions for Future Work

6.2.1 Extend Pattern Catalog

The current collection of safety patterns includes 15 high-level safety-related architectures for hardware or software systems which have so far been described in literature several times and which appear to be mature and also applied in practice. However, the current collection mostly describes redundancy-based architectures and could be extended to also include other safety patterns. For example, [RVK12] describe architectural safety patterns. Two examples for the described patterns are SEPARATED SAFETY and SAFETY OVERRIDES. SEPARATED SAFETY describes keeping the safety-critical part of a system to a minimum, which corresponds to the safety tactic *Simplicity*. SAFETY OVERRIDES explains that a safety system should be able to override values of less reliable systems to make sure that the output value is always safe. This pattern relates to the safety tactic *Override*. Such patterns are more specific aspects of the high-level architectures described in this thesis as patterns and could also be integrated into the pattern collection to give further guidance on how actual measures such as overrides or interlocks should be implemented. Other patterns which could also be used to extend the current pattern collection are presented in [RVK13a] and [RK13].

6.2.2 Towards a Pattern Language

The current collection presents the patterns in a uniform notation and also shows some connections between the patterns. However, these connections describe the topology of the pattern system which means that the connections describe how the structure of the different patterns is related. Usually, in pattern languages, the relations between the pattern give information about which patterns can or should be applied after another. However, the safety patterns described in this thesis are rather alternatives to one another and do not completely describe how to develop safety systems from beginning to end. Thus, there clearly is a lot of possible future work to include other safety patterns which describe solutions on a different level of abstraction (such as [RVK12], [RVK13a], or [RK13]) and to then find other pattern relations to these patterns such as *"can be applied next"* or *"is a prerequisite to applying the current pattern"*. Having such relations and having more safety patterns which also address safety issues on a different level of abstraction would make it possible to profit from the major benefit of the concept of patterns which is to present a good solution space for a specific domain (in this case for safety-critical systems) and to cover most occurring problems for that domain by providing sequences of pattern which can be applied to tackle a problem.

6.2.3 Security Certification

The collection of safety patterns focuses on safety systems and on how to reason on safety with respect to the IEC 61508 safety standard. From a security point of view, the patterns include relevant security threats. There is also potential to further investigate how the concept of linking the patterns to a standard can be utilized for the security domain. For example, as part of this thesis, the connection between security tactics and the Common Criteria security standard has been investigated (see **Publication 7**). This connection could be used to link security measures applied to mitigate security threats for a system to the Common Criteria standard and thus to also describe how the security goals of the system can be achieved by using methods from the security standard. However, that approach has not yet been fully integrated in the pattern-based development method described in this thesis and is thus left open for future work.

6.2.4 Evaluation

The thesis evaluates the patterns and their application by discussing their benefits during their use in an industrial project, by evaluating the appropriateness of the suggestions of IEC 61508 safety methods, by evaluating the impact of the patterns on security, and by qualitatively comparing the method to other pattern-based safety development methods. It would be interesting to also apply the patterns to other projects and to conduct an extended evaluation. Such an evaluation could include questionnaires regarding the application. In particular, feedback on the usefulness of the links between the IEC 61508 safety standard and the patterns would be interesting. This information could not be completely gathered with the industrial project described in this thesis, because the safety certification of the project was delayed and not completed at the point of writing this thesis.

A thorough evaluation of the pattern-based development method would describe the development of two identical systems with an identical development environment; one developed with and one without the method. In such an evaluation, the benefits of the method could be seen directly and perhaps measured. However, it is impossible to develop a system twice in completely identical environments and even making an approximation to such an identical setup (e.g. by having two teams develop the same system - one team with and one team without the method) would result in a huge financial overhead. That is the reason why such an extensive evaluation was not conducted as part of this thesis, even though it would be very important and would provide extremely interesting insights regarding the benefits and liabilities of the patterns and their application.

6.2.5 Tool Support

Development using the patterns could be made easier with the provision of appropriate tools. For example, a tool could store the patterns in a repository, let the user select an appropriate pattern, and then provide the information about the pattern including the safety and security GSN skeletons. The tool could provide the potential to modify and complete the GSN skeletons. Regarding safety, the tool could provide the IEC 61508 measures which are applicable for specific tactics applied in a pattern and the tool could provide logic regarding the STRIDE security analysis. This would be beneficial if additional elements have to be added to the pattern, which then also have to be considered from a security point of view.

Another example of a useful tool would be a tool which automatically generates the relations between the patterns as described in this thesis. The approach can be fully automated, but has been applied by hand to find the relations between the patterns. The reason for doing this was that with 15 patterns it was still manageable and implementing a tool would have been more time consuming. However, if more patterns were to be included into the pattern system, the usage of a tool would be very beneficial.

Chapter 7

Publications

This chapter provides publications by the author of this thesis. Figure 7.1 shows an overview of the publications and shows how the publications are related to address the overall topic of this thesis to build and apply safety patterns with security aspects also with respect to system certification. The publications in the upper part of the figure target specific parts of the methodology to build safety patterns with security aspects as presented in Chapter 3. The publications in the lower part of the thesis describe how to apply the patterns and how to evaluate the patterns and the pattern-based development process which is covered in Chapter 4 and Chapter 5.

The following first shows a list of the publications and the remainder of this chapter contains the full publications.

Publication 1: Preschern et al., *Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle*, Nordic Conference on Pattern Language of Programs (VikingPLoP), Ikaalinen, Finland, March 21st – 24th, 2013.
Publication 2: Preschern et al., *Building a Safety Architecture Pattern System*, 18$^{\text{th}}$European Conference on Pattern Language of Programs (EuroPLoP), Irsee, Germany, July, 10th – 14th 2013.
Publication 3: Preschern et al., *Security Analysis of Safety Patterns*, 20$^{\text{th}}$Conference on Pattern Language of Programs (PLoP), Monticello, USA, October, 23rd – 26th 2013.
Publication 4: Preschern et al., *Safety Architecture Pattern System with Security Aspects*, submitted to Transactions on Pattern Language of Programs, currently under review.
Publication 5: Preschern et al., *Pattern-Based Safety Development Methods: Overview and Comparison*, 19$^{\text{th}}$European Conference on Pattern Language of Programs (EuroPLoP), Irsee, Germany, July, 9th – 13th 2014.
Publication 6: Preschern et al., *Applying and Evaluating Architectural IEC 61508 Safety Patterns*, 5$^{\text{th}}$International Conference on Software Technology and Engineering (ICSTE), Bandar Seri Begawan, Brunei Darussalam, September, 28th – 29th 2013.
Publication 7: Preschern et al., *Catalog of Security Tactics linked to Common Criteria Requirements*, 19$^{\text{th}}$Conference on Pattern Language of Programs (PLoP), Tucson, USA, October, 19th – 21th 2012.
Publication 8: Preschern et al., *Quantitative Security Estimation based on Safety Architecture Design Patterns*, 3$^{\text{rd}}$International Conference on Software and Information Engineering (ICSIE), Singapore, Singapore, May, 1st – 2nd 2014.
Publication 9: Preschern et al., *Structuring Modular Safety Software Certification by Using Common Criteria Concepts*, 38$^{\text{th}}$Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Cesme, Turkey, September, 5th – 8th 2012.

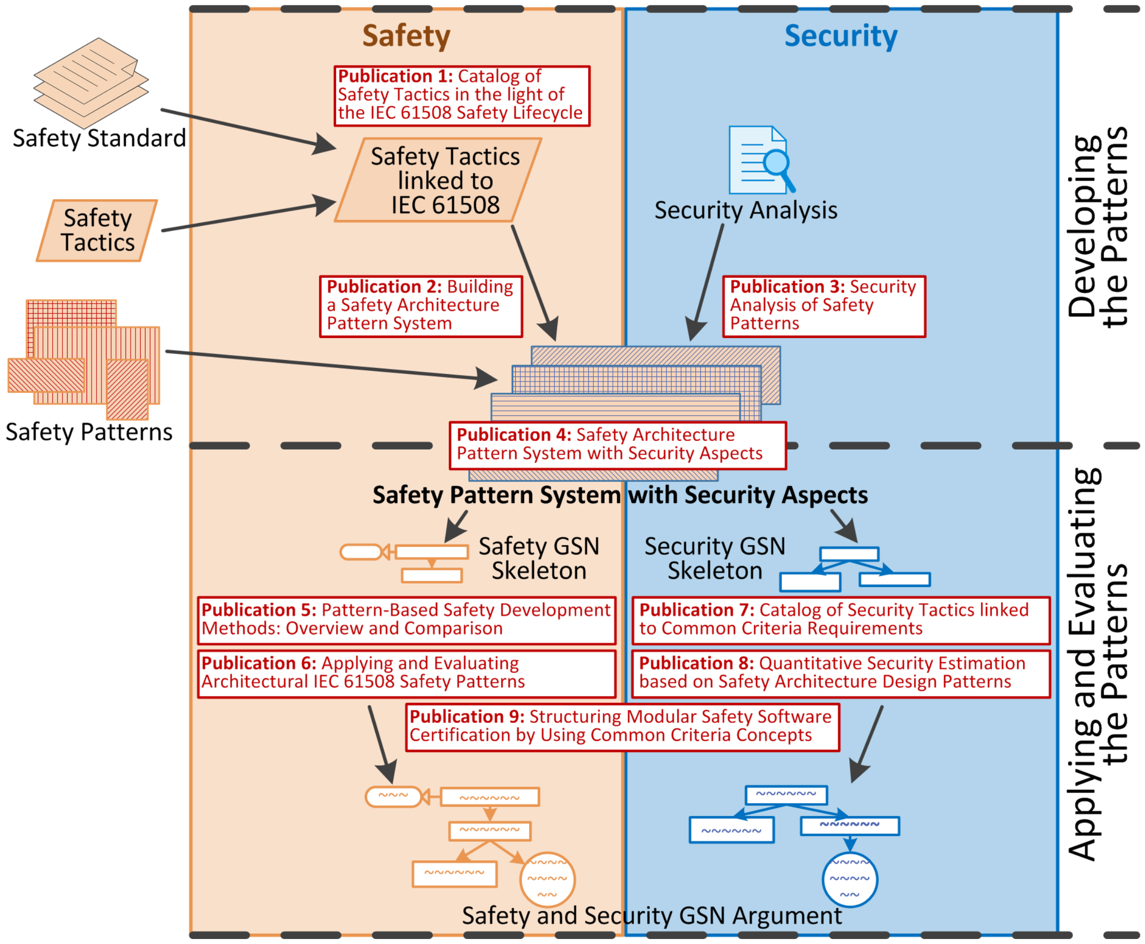

Safety
Security
Publication 1: Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle
Safety Standard
Safety Tactics linked to IEC 61508
Safety Tactics
Security Analysis
Publication 2: Building a Safety Architecture Pattern System
Publication 3: Security Analysis of Safety Patterns
Safety Patterns
Developing the Patterns
Publication 4: Safety Architecture Pattern System with Security Aspects
Safety Pattern System with Security Aspects
Safety GSN Skeleton
Security GSN Skeleton
Publication 5: Pattern-Based Safety Development Methods: Overview and Comparison
Publication 6: Applying and Evaluating Architectural IEC 61508 Safety Patterns
Publication 7: Catalog of Security Tactics linked to Common Criteria Requirements
Publication 8: Quantitative Security Estimation based on Safety Architecture Design Patterns
Publication 9: Structuring Modular Safety Software Certification by Using Common Criteria Concepts
Applying and Evaluating the Patterns
Safety and Security GSN Argument

Figure 7.1: Overview of the process describing the development, application, and evaluation of the safety patterns with security aspects. The figure includes relevant publications of the author of this thesis and shows which part of the process the publications address.

Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle

Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner

Institute for Technical Informatics, Graz University of Technology, Graz, Austria
christopher.preschern@tugraz.at, nermin.kajtazovic@tugraz.at, christian.kreiner@tugraz.at

Abstract. Safety tactics describe general architectural design decisions and their effect on the overall system safety. Currently these safety tactics do not directly address the consequences of design decisions on safety certification.
To establish this connection, we refine safety tactics by extracting information concerning architectural design decisions from the IEC 61508 safety standard. We generalize this information in order to describe the effect of safety tactic usage on different development phases of safety-critical systems. We provide the whole revised catalog of safety tactics and we show its application by analyzing the TRIPLE MODULAR REDUNDANCY design pattern regarding its safety tactic usage to evaluate the effect of the pattern on safety certification.

Keywords: safety tactics, IEC 61508

1 Introduction

Safety standards contain information about requirements which have to be fulfilled to achieve functional safety certification. Often some methods and architectures for fulfilling the safety requirements are suggested in the standard and in practice just these, sometimes outdated, methods and architectures are used. The introduction of new methods and architectures requires proof of their validity regarding functional safety which can be a tedious task and can increase certification costs significantly. There is no general evaluation of methods and architectures which allows to evaluate them regarding safety certification in order to aid the certification of novel concepts.

Safety patterns address this problem in a way that they describe the consequences of applying a specific architecture; however, they cover a rather specific and implementation focused view of this problem. To cope with the problem on a more general level, we evaluate the consequences of safety-related architectural design decisions (safety tactics) on safety certification. We examine existing safety tactics and discuss their suitability for the IEC 61508 safety standard. We mine architectures and methods suggested in the IEC 61508 standard regarding the tactics they use and regarding their effect on different phases of the safety lifecycle. Based on our analysis of used tactics in the IEC 61508 standard, we re-organize and re-structure existing safety tactics to be more intuitive. Furthermore, we refine the safety tactics by describing their influence on different safety lifecycle phases in general and more specific by relating IEC 61508 methods to the tactics. We present the refined catalog of safety tactics and we apply it to an example where we analyze the consequences of the TRIPLE MODULAR REDUNDANCY (TMR) pattern [1] on safety certification.

This paper is organized as follows. Section 2 gives an introduction to the IEC 61508 safety lifecycle and focuses on its realization phase which is later on analyzed for the tactics. Section 3 introduces the idea of tactics and Section 4 gives an overview of current tactics in the safety domain. Furthermore, in this Section we discuss why and how existing safety tactics should be modified. In Section 5 we present the tactic catalog with focus on the tactic influence on safety certification. Section 6 analyzes the TMR safety pattern by using the refined safety tactics. Section 7 gives an extended overview of related work on architectural tactics with focus on safety tactics. Section 8 concludes this work and gives an outlook on the future potential of this work.

2 IEC 61508 Safety Lifecycle

The safety lifecycle according to IEC 61508 provides a process framework which allows to achieve functional safety for a product by following the methods and requirements posed by the standard for each phase of the lifecycle. An overview of the lifecycle is shown in Figure 1.

The planning phases addressing the overall product safety include definition of concept and scope, a hazard and risk analysis resulting in safety requirements, and the allocation of Safety Integrity Levels (SILs) to components. During the planning phases, plans for the operation, maintenance, installation, and safety validation have to be defined. An important phase of the safety lifecycle is the product realization phase, which distinguishes between hardware and software implementation and can be divided into the following sub-phases:

- Requirements specification - Full specification of safety-related functions for the product, allocation of SILs to these functions, and specification of risk reduction measures for these functions.
- Validation planning - Preparation of a plan how to validate the system against the specified safety requirements.
- Design and development - Design and implementation of the safety-critical software/hardware according to the safety requirements.
- Integration - Integration/Assembly of developed software/hardware subsystems to form the complete safety-related product.
- Operation and maintenance - Activities to ensure the proper operation of the developed software/hardware product (does not cover system modifications).

In the phases after the realization of the system, the plans on operation, maintenance, installation, and safety validation have to be carried out. Additionally,

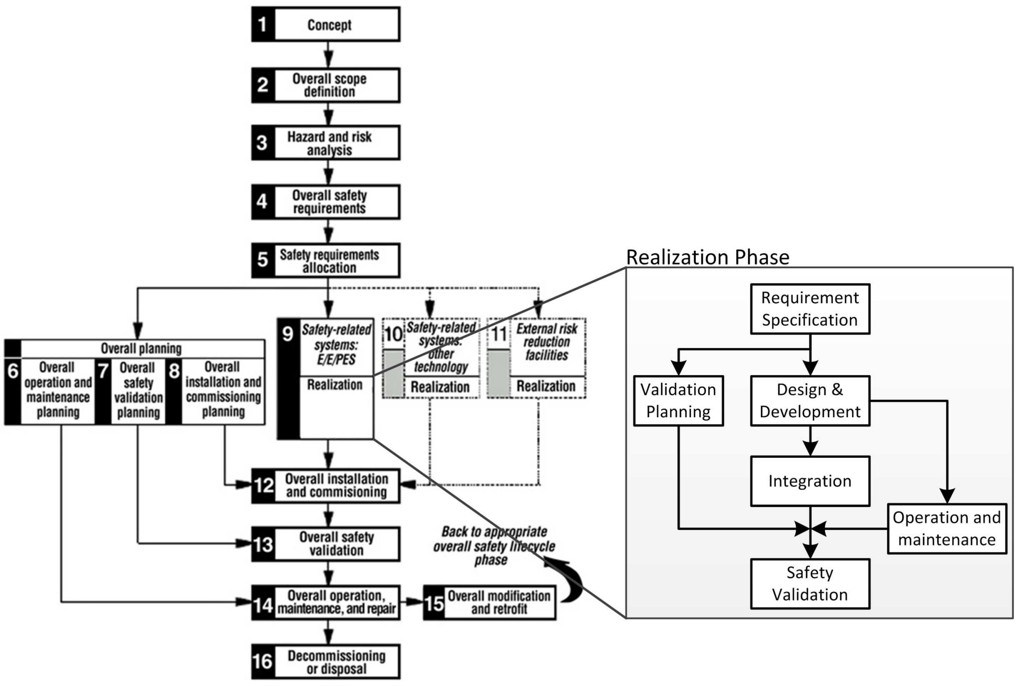

1 Concept
2 Overall scope definition
3 Hazard and risk analysis
4 Overall safety requirements
5 Safety requirements allocation
Overall planning
6 Overall operation and maintenance planning
7 Overall safety validation planning
8 Overall installation and commissioning planning
9 Safety-related systems: E/E/PES
Realization
10 Safety-related systems: other technology
Realization
11 External risk reduction facilities
Realization
12 Overall installation and commisioning
13 Overall safety validation
14 Overall operation, maintenance, and repair
15 Overall modification and retrofit
16 Decommissioning or disposal
Back to appropriate overall safety lifecycle phase
Realization Phase
Requirement Specification
Validation Planning
Design & Development
Integration
Operation and maintenance
Safety Validation

Fig. 1. IEC 61508 safety lifecycle (incl. realization phase) [2]

other phases of the lifecycle address product modifications, decommissioning, and disposal.

In this paper we focus on the effects of architectural safety tactics on the product realization phase and the following phases like safety validation and product modification. We do not describe the effect of architectural safety tactics on all of the phases mentioned above, because when analyzing the safety standard, we did not find relationships of the tactics to all of the safety lifecycle phases, especially not to the early phases.

3 Introducing Tactics

Tactics are architectural design decisions which influence and manipulate quality attributes [3]. Compared to design patterns, they describe general concepts or principles and do not describe solutions for a problem in a given context. For example, the Voting tactic describes how to achieve failure containment by choosing an appropriate system output from redundant system components. Compared to patterns, the tactic is more general and does not describe a specific solution but rather provides the underlying idea for possible solutions. In this case, a possible solution could be the TMR pattern which uses the Voting tactic to choose for the majority of three redundant subsystem outputs. Usually, a tactic can be found in several architectures or patterns and can even be seen as building blocks for design patterns [4].

It is difficult to keep tactics and patterns apart as there is no clear boarder between the two. However, Ryoo et al. [5] specify some criteria to identify tactics. For a design decision on order to be a tactic, it has to be atomic. This means that it cannot be divided into other multiple tactics, however it can be refined. For example, the Redundancy tactic is refined by the Replication Redundancy tactic and the Diverse Redundancy tactic, but it is not composed of them. Furthermore, Ryoo et al. say that tactics focus on a single quality attribute (e.g. safety) and patterns usually affect several quality attributes.

4 Safety Tactic Catalog

A collection of safety tactics presented by Wu [6] is shown in Figure 2. These tactics were mined from safety architectures described in literature and address failure avoidance, failure detection, and failure containment.

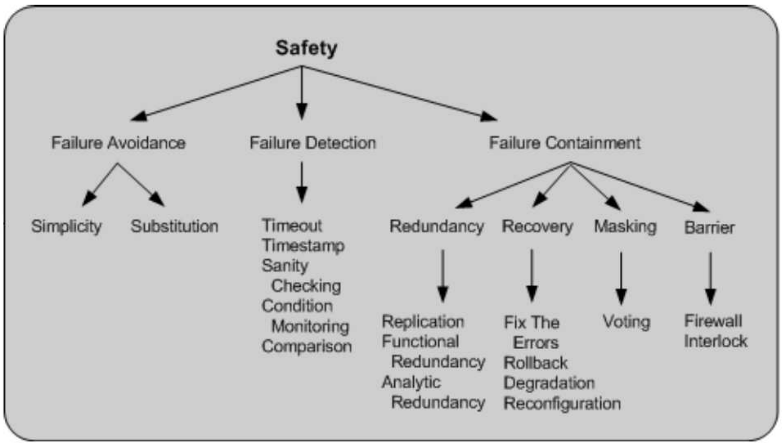


Fig. 2. Safety tactics proposed by Wu [6] (arrows show tactic refinements)

We analyzed methods and architectures used in the IEC 61508 standard and related them to Wu's safety tactics. Part 7 of the IEC 61508 standard explains several safety-related methods and architectures and describes their aims. We linked them to Wu's safety tactics by manually searching for similarities between the method or architecture aims and the tactic aims.

For some tactics we could not find any relationship to the standard at all and some methods and architectures had very similar relationships to the same set of tactics indicating that these tactics are rather similar. Furthermore, some of the tactics describe rather specific safety-related solutions (e.g. Timestamp), while others describe general concepts (e.g. Voting). This motivated us to revisit the safety tactics, to make the safety tactic catalog more intuitive. We add tactics we found rather often in the IEC 61508 standard to the catalog and we skip

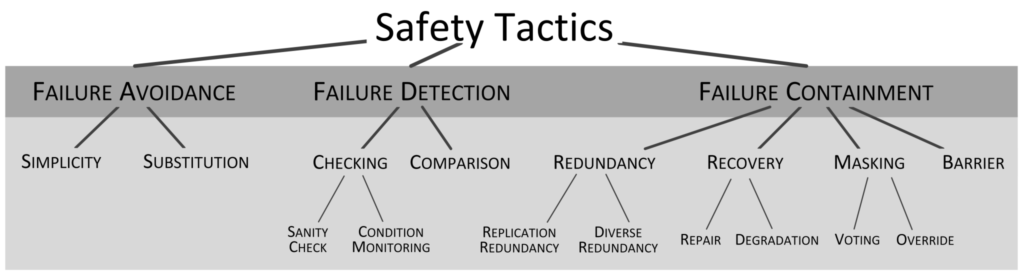


Fig. 3. Re-organized safety tactics (arrows show tactic refinements)

tactics which we did not find in the standard or which were very implementation-specific.

The detailed description of each tactic we use in the following section and the detailed process how we manually analyzed the IEC 61508 standard will be given later on in Section 5.

4.1 Re-organized Safety Tactics

Figure 3 shows our re-organized safety tactics catalog. We keep Wu's general categorization of safety tactics in failure avoidance, failure detection, and failure containment tactics. We do not modify the failure avoidance tactics, because methods regarding failure avoidance in the IEC 61508 standard could perfectly well be mapped to Wu's tactics. However, we change parts of the failure detection and failure containment tactics as explained in the following.

Wu's safety tactics Sanity Check and Condition Monitoring check a system state or value against additionally introduced redundant information. The difference is, that Sanity Check introduces this information in the specification, while Condition Monitoring introduces the information in the implementation phase. Due to the similarities of the two tactics, we generalize them in a Checking tactic. A similar tactic was already suggested in [7] where Wu's safety tactics were also slightly adapted.

We recognized that just very few IEC 61508 methods used the Timestamp or Timeout tactic. This lead to the idea that they might be rather specific tactics and not very general. The Timeout tactic detects excessive time-resource usage. This is a simple check of the time condition compared to a specified limit and can be considered as a Sanity Check. The Timestamp tactic checks the validity of an entity by checking a timestamp attached to it, which also is a Sanity Check of a beforehand specified time condition. We therefore see the Timeout and Timestamp tactics as refinements of Sanity Check; however, we do not include them in our tactic collection because we want to focus on more general tactics. We are not the first to eliminate Timestamp and Timeout from the safety tactics collection; also in [7] these tactics are omitted.

Wu distinguishes between three types of redundancy: replication (redundant identical hardware), functional (redundant implementation), and analytical (redundant specification). The methods from the safety standard which we linked to functional and analytical redundancy are very similar. Therefore, we combine these two types of redundancy and call it Diverse Redundancy.

No methods of the IEC 61508 standard were mapped to Wu's Rollback or Reconfiguration tactics, because they rather address availability concerns which are covered by availability tactics [8]. However, several parts of the standard suggest recovery from errors by Repair and Degradation which we include in our tactic catalog.

We added the Override tactic to Masking, because the safety standard often describes fail-safe mechanisms, which differ from the Voting tactic. These mechanisms are based on output decisions of redundant channels where a specific output state (safe state) is preferred to other states.

The Interlock and Firewall tactic are very implementation-specific and similar mechanisms are described in literature as patterns (Output Interlock pattern [9], Firewall pattern [10]). Therefore, we omit these tactics.

5 Refined Tactics Catalog

In this section we present the catalog of safety tactics and discuss their consequences on different phases of the safety lifecycle. We structure each tactic into the sections **Aim**, **Description**, **Influence on the Safety-Lifecycle**, and **Related IEC 61508 Methods**. We refine Wu's safety tactics mostly with information from the seven parts of the IEC 61508 standard [2]. We studied the standard to find links between parts of the standard and the safety tactics. We started with part 7 of the standard which contains a collection of techniques which are often applied in the safety domain. We mapped these techniques to the safety tactics by finding similarities between the technique aims and the tactic aims. The techniques serve as the main source for the **Related IEC 61508 Methods** section of the tactics. We also generalized information about the description and the aim of the techniques to refine the **Description** and **Aim** sections of Wu's tactics.

The techniques of part 7 are often referenced in other parts of the standard, especially often in parts 2 and 3. We analyzed the context of these references to find out further information about the tactics and their effects on different parts of the safety lifecycle. From the safety lifecycle described in Section 2, we just present the phases directly influenced by the safety tactics, which are: Specification, Design and Development, Integration, Operation and Maintenance, Modification, Verification, and Safety Validation. The effect of tactics on these parts of the safety lifecycle is given in the **Influence on the Safety-Lifecycle** section of the tactics and is mainly based on the parts 2 and 3 of the safety standard.

Additionally to the above mentioned approach to mine the IEC 61508 standard for safety tactics, we also went through the parts 1-6 of the standard again from the beginning to the end to find any connections to the safety tactics. This yielded a very similar result to the above mentioned approach. It only differed in a few additional connections between the standard and the tactics mostly coming from part 6.

Now we present the refined safety tactic catalog.

5.1 Failure Avoidance

SIMPLICITY and SUBSTITUTION are failure avoidance tactics. If applicable, they are often preferred to failure detection and failure containment tactics [11], because they are rather independent from other tactics and do not create overhead for other safety lifecycle phases.

SIMPLICITY

Aim - Avoid failures through keeping the system as simple as possible.

Description - SIMPLICITY reduces the system complexity. It includes structuring methods or cutting unnecessary functionality and organizes system elements or reduces them to their core safety functionality, thus, eliminating hazards. An example for the application of the SIMPLICITY tactic is an emergency stop switch system which is usually kept as simple as possible.

Influence on the Safety-Lifecycle - The tactic reduces effort for every phase in the safety lifecycle due to reduced system complexity or even reduced system functionality. However, most other safety tactics contradict SIMPLICITY, because they require additional system components (e.g. a voter) which are not absolutely necessary for the core system functionality. In particular for early phases SIMPLICITY enables significant complexity reduction. When applied during the specification phase, it increases understandability and predictability of the system behavior (IEC 61508-3 Annex F). For the Design&Development phase, it enables easier system development which is required in IEC 61508-3 7.4.2.2, 7.4.3.6, 7.4.2.6 and 7.6.2.2. However, the tactic might also put constraints on system development. For example, IEC 61508-3 7.4.4.13 requires to limit the programming language command set to the usage of safe, well-proven commands.

Related IEC 61508 Methods - IEC 61508-7: B.2.1 structured specification, B.3.2 structured design, C.2.7 structured programming, E.3 structured description method, C.4.2 programming language subset, C.4.2 limit asynchronous constructs, E.5.13 software complexity controller

SUBSTITUTION

Aim - Avoid failures though usage of more reliable components.

Description - Components or methods are replaced by other components or methods one has higher confidence in. For hardware and software this can mean usage of existing components which are well-proven in the safety domain.

Influence on the Safety-Lifecycle - Changing software or hardware components can require re-doing the safety hazard analysis [6]. However, software components can also be exchanged with previously developed components or third-party components to reduce the certification effort by re-using certification knowledge or documents for these components. SUBSTITUTION can increase hardware or third-party component costs if safer components are used. For example, buying a SIL3 component usually is more expensive than buying a SIL2 component.

Related IEC 61508 Methods - IEC 61508-7: B.3.3 usage of well-proven components, B.5.4 field experience, C.2.10 usage of well-proven/verified software elements, E.20 application of validated soft-cores, E.35 application of validated hard-cores, E.41 usage of well-tried circuits, C.4.3 certified tools and compilers, C.4.4 well-proven tools and compilers, E.4 well-proven tools, E.42 well-proven production process, E.28 application of well-proven synthesis tools, E.29 application of well-proven libraries

5.2 Failure Detection

Every failure detection method requires some kind of redundancy and testing of the redundant information. The CHECKING tactics introduce diverse information to check a system and the COMPARING tactic compares fully redundant information or systems.

CHECKING - SANITY CHECK

Aim - Detection of implausible system outputs or states.

Description - The SANITY CHECK tactic checks whether a system state or value remains within a valid range which can be defined in the system specification or which is based on knowledge about the internal structure or nature of the system. An example for a SANITY CHECK is a stuck-at fault RAM-test which checks the proper functionality of the memory during system runtime. The test is based on the understanding of the memory behavior (if we write data to the memory, we should later on be able to read the same data). Faults are detected if the memory behaves differently.

Influence on the Safety-Lifecycle - Plausible system outputs and states have to be specified (e.g. IEC 61508-3 C.2 3a where preconditions limit the system input range). This value range limitation can help during the system verification, because just the defined value range has to be tested (IEC 61508-3 C.2). For safety validation it can be argued that the SANITY CHECK introduces a diverse implementation for checking the safety functionality and therefore detects random as well as systematic implementation or design faults to some extent (IEC 61508-6 D.1.4).

Related IEC 61508 Methods - IEC 61508-7: A.1.2 monitoring relay contacts, A.2.7 analog signal monitoring, A.3.1-A.3.3 self-tests, A.4.1-A.4.4 checksums, A.5.1-A.5.5 RAM-Tests, A.6.1 test pattern, A.7.1 one-bit hardware redundancy, A.7.2 multi-bit hardware redundancy, A.7.4 inspection using test patterns, A.9 temporal and logical program monitoring, C.3.3 assertion programming, C.5.3 interface checking, C.4.1 strong typed programming language

Checking - Condition Monitoring

Aim - Detect deviations from the intended system outputs or states.

Description - Condition Monitoring checks whether a system value remains within a reasonable range compared to a more reliable, but usually less accurate, reference value. The reference value is computed at runtime by a redundant part in the implementation which can be based on system input values and is not pre-known from the specification (like it would be the case for Sanity Check). An example for Condition Monitoring is a system which has to be time-synchronized via the Internet and which checks if the synchronized time is feasible by comparing it to an internal clock.

Influence on the Safety-Lifecycle - An additional element providing the reference value has to be implemented. In general, the Condition Monitoring tactic implies more development overhead than Sanity Check. Condition Monitoring primarily protects from random faults. However, if it uses a diverse implementation for monitoring the safety functionality, also systematic implementation or design faults can be detected (IEC 61508-6 D.1.4).

Related IEC 61508 Methods - IEC 61508-7: A.1.1 failure detection by online monitoring, A.6.4 monitored outputs, A.8.2 voltage control, A.9 temporal and logical program monitoring, A.12.1 reference sensor, A.13.1 monitor

Comparison

Aim - Detection of discrepancies of redundant system outputs.

Description - Comparison tests if the outputs of fully redundant subsystems are equal in order to detect failures. The Comparison tactic usually implies the usage of a redundancy tactic. An example for the application of the Comparison tactic is a dual-core processor running in lock-step mode. The processor runs the same software on both cores and compares their outputs after each cycle.

Influence on the Safety-Lifecycle - An additional element which requires resources at runtime to compare the subsystems has to be implemented.

Related IEC 61508 Methods - IEC 61508-7: A.1.3 comparator, A.6.5 input comparison/voting

5.3 Failure Containment

Failure containment describes ways how to handle failures which are recognized by failure detection. The Masking and Barrier tactics prevent failures from affecting other parts of the system and the Recovery tactics deals with correcting failures. The Redundancy tactics provide multiple systems which are necessary for some other tactics.

Redundancy - Diverse Redundancy

Aim - Introduction of a redundant system which allows detection or masking of failures in the specification or implementation as well as random hardware failures.

Description - Diverse Redundancy can be applied to the specification or to the implementation level. In a system using Diverse Redundancy on the implementation level, redundant components use different implementations which were developed independently from the same specification. Diverse Redundancy on a specification level goes one step further and additionally requires that even the requirement specifications for the redundant components have to be set up by individual teams.

Influence on the Safety-Lifecycle - Diverse Redundancy highly contradicts the Simplicity tactic, because the additionally introduced redundant systems require a lot of effort (multiple effort for specification, implementation, verification, modification, ...) which does not add to the system functionality. If redundant systems are used, then it has to be shown for safety validation that the systems are independent from each other which can be achieved by application of the Barrier tactic (IEC 61508-1 7.6.2.7). Redundant hardware systems can more easily be validated for safety, because for a system with no hardware fault tolerance, diagnostic tests have to be run each time before computing a safety-critical function. This requirement is not so strict for hardware redundant systems (IEC 61508-2 7.4.4.1.4, 7.4.4.1.5, 7.4.4.2.1, 7.4.5.3).

Related IEC 61508 Methods - IEC 61508-7: A.7.6 information redundancy, A.13.2 cross-monitoring of multiple actuators, B.1.4 diverse hardware, C.4.4 diverse programming

Redundancy - Replication Redundancy

Aim - Introduction of a redundant systems which allows detection or masking of random hardware failures (not systematic failures).

Description - Replication Redundancy means introduction of a redundant system of the same implementation. The redundant systems maintain the same functionality, use identical hardware, and run the same software implementation. An example for Replication Redundancy is the RAID1 data storage technology.

Influence on the Safety-Lifecycle - Replication Redundancy requires multiple effort for hardware installation and modification. If redundant systems are used, then it has to be shown for safety validation that the systems are independent from each other which can be achieved by application of the Barrier tactic (IEC 61508-1 7.6.2.7). Redundant hardware systems can more easily be validated for safety, because for a system with no hardware fault tolerance, diagnostic tests have to be run each time before computing a safety-critical function. This requirement is not so strict for hardware redundant systems (IEC 61508-2 7.4.4.1.4, 7.4.4.1.5, 7.4.4.2.1, 7.4.5.3).

Related IEC 61508 Methods - IEC 61508-7: A.2.1 tests by redundant hardware, A.2.5 monitored redundancy, A.3.5 reciprocal comparison by software, A.4.5 block replication, A.6.3 multi-channel output, A.7.3 complete hardware redundancy, A.7.5 transmission redundancy

Recovery - Repair

Aim - Bring a failed system back to a state of full functionality.

Description - The full system functionality is manually or automatically restored if a system failure occurs.

Influence on the Safety-Lifecycle - A Repair or Degradation tactic is necessary for all non-redundant hardware elements which maintain a safety functionality (IEC 61508-2 7.4.8.2). However, complex recovering systems like self-reconfiguring systems are not recommended by the standard (IEC 61508-3 A.2) and make validation more complicated.

Related IEC 61508 Methods - IEC 61508-7: C.3.9 error correction, C.3.10 dynamic reconfiguration

Recovery - Degradation

Aim - Degradation brings a system with an error into a state with reduced functionality in which the system still maintains the core safety functions.

Description - Degradation systems define a core safety functionality. The systems maintain this safety functionality and additional non-critical functions. In case of an error, the system falls back into a degraded mode in which it just maintains the core safety functionality. An example where the Degradation tactic is often applied are automation systems. These systems control safety-critical processes and often visualize these processes in a GUI. If the system has too few resources (e.g. processing time), the system stops the GUI service and just focuses on its core functionality to control the safety-critical processes.

Influence on the Safety-Lifecycle - Degradation mechanisms for the system have to be specified (IEC 61508-2 7.2.3.2) and a Repair or Degradation tactic is necessary for all non-redundant hardware elements which maintain a safety functionality (IEC 61508-2 7.4.8.2). Degradation can decrease the safety validation effort, because just the degradation mechanism and the core safety functionality have to be validated. Additionally, the tactic fulfills the requirement of the standard to describe a well defined behavior in case of errors (IEC 61508-2 7.2.3.2, IEC 61508-3 7.2.3.2).

Related IEC 61508 Methods - IEC 61508-7: A.8 voltage supply error handling, C.3.8 degraded functions

Masking - Voting

Aim - Mask the failure of a subsystem so that the failure does not propagate to other systems.

Description - Voting makes a failure transparent. The tactic does not try to repair the failure, but it hides the failure through choosing a correct result from redundant subsystems. It decides for the majority of the output values.

Influence on the Safety-Lifecycle - In order to apply Voting, a redundancy tactic has to be used and a voter element has to be implemented. Subsystems of a voting system can be repaired while in operation, because the overall system can still operate if a subsystem is under repair (IEC 61508-6 B.3.1). However, voting systems are not as safe as systems which just compare their results and ensure a safe state if any of the results differs (IEC 61508-6 B.3).

Related IEC 61508 Methods - IEC 61508-7: A.1.4 voter, A.6.5 input comparison/voting

MASKING - OVERRIDE

Aim - Mask the failure of a subsystem so that the failure does not propagate to other systems.

Description - The OVERRIDE tactic forces the system output to a safe state. For example, if we have a system which is in a safe state when shut off, we can apply the OVERRIDE tactic to shut off the system if we have doubt about the system output (e.g. if an output validity check fails). In this scenario overriding the system output with a safe output value decreases the availability of the system. Another form of the OVERRIDE tactic, which does not decrease the availability and is closely related to the VOTING tactic, chooses the output of redundant subsystems by preferring one subsystem or one output state over another.

Influence on the Safety-Lifecycle - A preferred system output state has to be defined and an override mechanism has to be implemented. OVERRIDE systems are easier to validate, because they follow the fail-safe principle (IEC 61508-1 7.10.2.6).

Related IEC 61508 Methods - IEC 61508-7: A.1.3 comparator, A.1.5 idle current principle, A.6.5 input comparison/voting, A.8.1 overvoltage protection with safety shut-off, A.8.3 power-down with safety shut-off

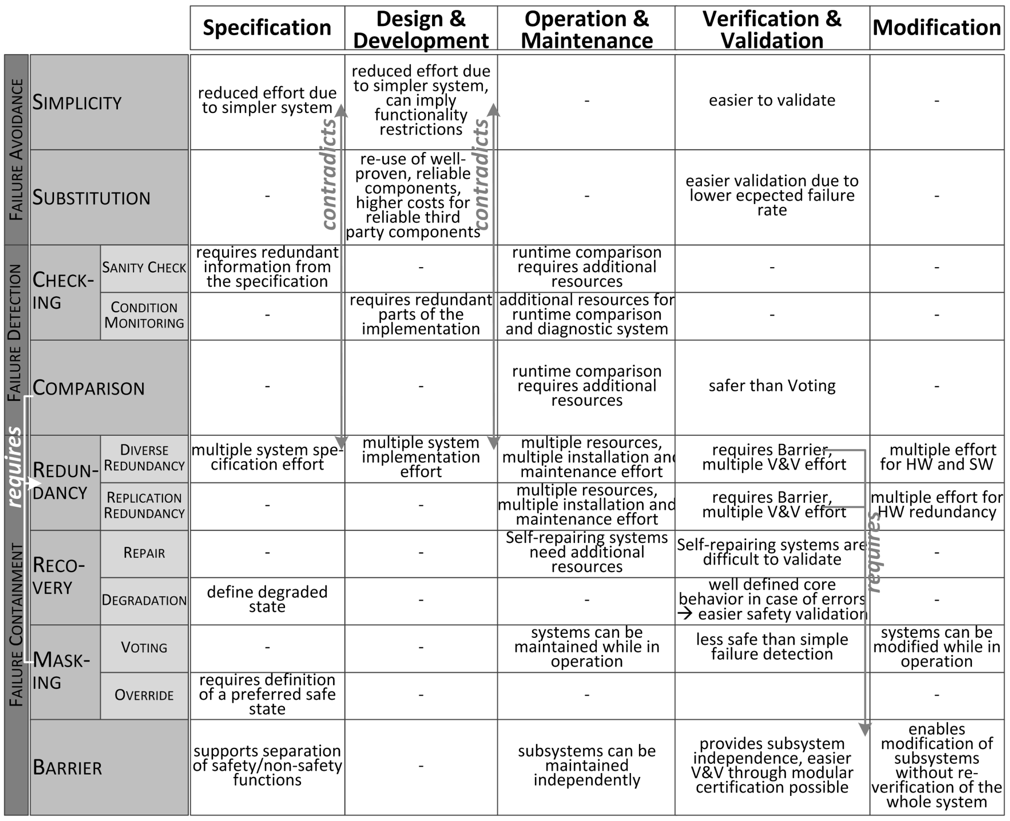
BARRIER

Aim - Protect a subsystem from influences or influencing other subsystems.

Description - The BARRIER tactic provides a mechanism to protect from unintentional influences between subsystems. To apply BARRIER, the interfaces between subsystems have to be analyzed and specified. These interfaces are controlled at runtime by a trustworthy component (the BARRIER) which often is an already existing reliable mechanism. An example for a BARRIER is a memory protection unit which controls and restricts the communication between different tasks.

Influence on the Safety-Lifecycle - The interfaces between subsystems have to be specified. According to IEC 61508-3 8.3.1, non-safety related functions should be separated from safety-related functions, which can be achieved by the BARRIER tactic. It can also aid the SIMPLICITY tactic by structuring the system (IEC 61508-2 7.2.2.1). BARRIER enables modular safety certification and modification and can reduce the validation effort if it is proven that the subsystems cannot unintentionally influence each other which has to be shown by an effect analysis (IEC 61508-3 C.8, Annex F).

Related IEC 61508 Methods - IEC 61508-7: A.11 separation of energy lines from information lines, B.1.3 separation of safety functions from non-safety functions, B.3.4 modularization, C.2.8 information hiding/ encapsulation, C.2.9 modular approach, E.12 modularization, C.3.11 time-triggered architecture


contradicts
contradicts
requires
requires

			Specification	Design & Development	Operation & Maintenance	Verification & Validation	Modification
Failure Avoidance	Simplicity		reduced effort due to simpler system	reduced effort due to simpler system, can imply functionality restrictions	-	easier to validate	-
Failure Avoidance	Substitution		-	re-use of well-proven, reliable components, higher costs for reliable third party components	-	easier validation due to lower ecpected failure rate	-
Failure Detection	Checking	Sanity Check	requires redundant information from the specification	-	runtime comparison requires additional resources	-	-
Failure Detection	Checking	Condition Monitoring	-	requires redundant parts of the implementation	additional resources for runtime comparison and diagnostic system	-	-
Failure Detection	Comparison		-	-	runtime comparison requires additional resources	safer than Voting	-
Failure Containment	Redundancy	Diverse Redundancy	multiple system specification effort	multiple system implementation effort	multiple resources, multiple installation and maintenance effort	requires Barrier, multiple V&V effort	multiple effort for HW and SW
Failure Containment	Redundancy	Replication Redundancy	-	-	multiple resources, multiple installation and maintenance effort	requires Barrier, multiple V&V effort	multiple effort for HW redundancy
Failure Containment	Recovery	Repair	-	-	Self-repairing systems need additional resources	Self-repairing systems are difficult to validate	-
Failure Containment	Recovery	Degradation	define degraded state	-	-	well defined core behavior in case of errors → easier safety validation	-
Failure Containment	Masking	Voting	-	-	systems can be maintained while in operation	less safe than simple failure detection	systems can be modified while in operation
Failure Containment	Masking	Override	requires definition of a preferred safe state	-	-		-
Failure Containment	Barrier		supports separation of safety/non-safety functions	-	subsystems can be maintained independently	provides subsystem independence, easier V&V through modular certification possible	enables modification of subsystems without re-verification of the whole system

Fig. 4. Safety tactics and their effect on different phases of the safety lifecycle

5.4 Overview and Discussion of the Safety Tactics

Figure 4 gives an overview of our re-organized safety tactics and their influence on the safety lifecycle and presents relationships between the tactics. The information is mostly based on the *Influence on the Safety-Lifecycle* parts of the tactics described in the previous section.

The revised version of the safety tactics provides more consistency compared to Wu's tactics. The problem with Wu's Timestamp and Timeout tactic as special case of Sanity Check is resolved.

Just few methods and architectures from the IEC 61508 standard address failure containment tactics. We think that the reason why just few failure containment tactics were found in the safety standard is that some of the tactics, such as Masking for example, are more concerned with availability than with safety. Therefore the standard does not focus on these tactics.

6 Refining the TMR Pattern by Reasoning with Tactics

In this section we use our refined safety tactics to discuss the safety-related effects of applying the TMR architectural pattern.

The TMR architecture shown in Figure 5 uses three channels and compares the outputs of the channels. A voter decides for the output value which is given by at least two of the channels. The architecture therefore allows one channel to be erroneous while still maintaining full system functionality. In our example we assume simple hardware replication with identical software running on the channels.

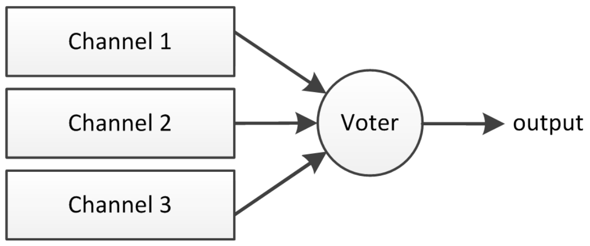


Fig. 5. Homogeneous TMR architecture

The TMR architecture described above uses two general safety tactics: Redundancy and Masking. More specific, Replication Redundancy is used, because there are identical redundant channels and Voting is used to mask errors of a single channel. If we have a look at Figure 4, we can see that the Replication Redundancy tactic requires the Barrier tactic during the Verification&Validation phase of the safety lifecycle. This means that to design a TMR system in the safety domain, also the Barrier tactic has to be considered right at the beginning of the architecture design in order to assure that the three subsystems do not influence each other in terms of common cause failures. This information might be obvious to a safety domain expert, however, for unexperienced system architects such information can be crucial.

	Specification	D&D	Operation & Maintenance	Verification & Validation	Modification
Replica-tion	-	-	multiple resources, multiple installation and maintenance effort for HW	requires Interlock, multiple V&V effort	multiple effort for HW redundancy
Voting	-	-	systems can be maintained while in operation	less safe than simple failure detection	systems can be modified while in operation
Barrier	supports separation of safety/non-safety functions	-	subsystems can be maintained independently	provides subsystem indepen-dence, easier V&V through modular certification possible	enables modification of sub-systems without re-verifi-cation of the whole system

Table I: Safety tactics for the homogeneous TMR architecture

We end up with three tactics which are used by the TMR system: Replication Redundancy, Voting, and Barrier. Table I shows the TMR relevant tactics taken from Figure 4. We can see that our TMR architecture influences the Operation&Maintenance phase in a way that multiple hardware is required and has to be installed. This implies multiple hardware maintenance effort. However, the three hardware channels can be maintained independently and they can even be maintained during operation due to the BARRIER and VOTING tactics. For safety validation, random hardware channel failures are independent and systematic failures are not detected. The VOTING tactic requires validation of the correct functionality in case of an error and is therefore more difficult to validate than simple systems which shut down or go to a safe state in case of errors. Just like with maintenance, hardware modifications for single channels can be done during operation. Multiple effort is required if the modification affects the redundant channels. The effort for system specification and development is not increased due to the simple usage of replication.

If we look at the detailed tactic descriptions from Section 5, we can get further information for the TMR pattern in terms of a quick reference to the IEC 61508 standard. As one example, the REPLICATION REDUNDANCY tactic is connected to IEC 61508-2 7.4.4.1.4 which says that self-tests for a single channel do not have to be executed each time before the execution of a safety function if redundant channels are present. It is sufficient to execute the self-tests once a day. Such quick references provide us with very detailed IEC 61508 related information.

The evaluation of the TMR pattern through the usage of our refined set of safety tactics leads to much more detailed information regarding safety, in particular safety certification, than existing safety pattern catalogs such as [12] offer.

7 Related Work on Safety Tactics

In this section we present related work on architectural tactics with focus on safety tactics. We also present patterns which are related to the IEC 61508 standard.

Bachmann et al. introduce the idea of architectural tactics and describe their relation to system quality attributes [3]. They present a collection of tactics for availability, security, testability, usability, modifiability, and performance. Wu and Kelly extended this collection by adding a set of tactics for the safety quality attribute [6] [13]. They further develop an approach how to apply safety tactics by stating anti-requirements which can be handled by the application of safety tactics [14]. This approach is explained in more detail in [15], where a whole architectural safety-reasoning framework is presented.

Another approach of how to reason about the usage of safety tactics is presented in [16] and [11], where safety attributes of a system are evaluated by risk-based qualitative reasoning. This reasoning is done before and after the application of a safety tactic in order to evaluate the applicability of the tactic. The application of safety tactics in order to build a safe architecture is also described in [17] and [7] with focus on the integration of the tactics into the V-model which is commonly used for IEC 61508 system development.

To the best of our knowledge there is no work directly relating safety tactics to a safety standard so far, however, Armoush [12] constructs an extensive catalog of safety patterns and evaluates them regarding IEC 61508 safety certification by presenting the applicability of a pattern according to recommendations in the safety standard. Compared to our work he does not discuss the influence on the safety system over the whole safety lifecycle and does not give much detail regarding the influence on safety certification. [18] covers organizational patterns for IEC 61508 software development. They focus on patterns for software development and not on the relation of IEC 61508 to architectural patterns.

8 Conclusion

In this paper we provide a revised catalog of safety tactics and relate these tactics to the IEC 61508 safety standard. This allows us to evaluate generic architectures like safety patterns regarding their effect on safety certification during different phases of the safety lifecycle. With the connection between safety tactics and the IEC 61508 standard it is now easier to provide a system architect with information about the safety related consequences of choosing a specific tactic or pattern. Here, an advantage of the safety tactics is that compared to the safety standard, they provide system architects with a view of the safety domain, which is more familiar to them. The tactic catalog therefore provides a good source of information for early architectural decisions for systems which have to be safety certified.

The re-organized set of safety tactics can serve as a basis for future work on refining patterns in the safety domain. Future work could also include refining our tactics or evaluating them with respect to a different safety standard. We believe that our re-organized version of safety tactics builds a mature set of safety tactics and that system developers can use them to argue for the safety of their system during safety certification.

ACKNOWLEDGMENTS

We would like to thank our shepherd Jari Rauhamäki who significantly helped to improve this paper. He provided us with good overall feedback and in particular helped to improve the tactic descriptions of this paper with his detailed safety-related knowledge.

References

1. Douglass, B.P.: Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Pearson (2002)
2. International Electrotechnical Commission: IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems (2010)

3. Bachmann, F., Bass, L., Klein, M.: Deriving Architectural Tactics : A Step Toward Methodical Architectural Design. Technical Report March, Carnegie Mellon Software Engineering Institute (2003)
4. Kumar, K., Prabhakar, T.V.: Pattern-oriented Knowledge Model for Architecture Design. In: 17th Conference on Pattern Languages of Programs (PLoP). (2010)
5. Ryoo, J., Laplante, P., Kazman, R.: A Methodology for Mining Security Tactics from Security Patterns. In: 2010 43rd Hawaii International Conference on System Sciences, IEEE (2010) 1–5
6. Wu, W.: Safety Tactics for Software Architecture Design. Master's thesis, The University of York (2003)
7. Hill, A., Nicholson, M.: Safety tactics for reconfigurable process control devices. In: 4th IET International Conference on Systems Safety 2009. Incorporating the SaRS Annual Conference, IET (2009)
8. Bass, L., Clements, P., Rick: Software Architecture in Practice. 2 edn. Addison-Wesley (2003)
9. Rauhamäki, J., Kuikka, S.: Patterns for control system safety. In: Proceedings of the 18th European Conference on Pattern Languages of Programms (EuroPLoP '2013). (2013)
10. Schumacher, M.: Firewall Patterns. In: Proceedings of the 8th European Conference on Pattern Languages of Programms (EuroPLoP '2003), Universitaetsverlag Konstanz (2003)
11. Im, T.: A Reasoning Framework for Dependability in Software Architectures. PhD thesis, Clemson University (2010)
12. Armoush, A.: Design patterns for safety-critical embedded systems. PhD thesis, RWTH Aachen University (2010)
13. Wu, W., Kelly, T.: Safety tactics for software architecture design. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC), IEEE (2004) 368–375
14. W. Wu and T. Kelly: Managing Architectural Design Decisions for Safety- Critical Software Systems. In: Quality of Software Architectures, Springer (2006) 59–77
15. Wu, W.: Architectural Reasoning for Safety- Critical Software Applications. PhD thesis, University of York (2007)
16. Im, T., Vullam, S., McGregor, J.: Reasoning about Safety during Software Architecture Design. In: Proceedings of the 19th International Conference on Software Engineering and Data Engineering (SEDE 2010). (2010)
17. Hill, A.: Safety Tactics for Reconfigurable Process Control Devices. Master's thesis, University of York (2008)
18. Vuori, M., Virtanen, H., Koskinen, J., Katara, M.: Safety Process Patterns in the Context of IEC 61508-3. Technical report, Tampere University of Technology (2011)

Building a Safety Architecture Pattern System

CHRISTOPHER PRESCHERN, Institute for Technical Informatics, Graz University of Technology
NERMIN KAJTAZOVIC, Institute for Technical Informatics, Graz University of Technology
CHRISTIAN KREINER, Institute for Technical Informatics, Graz University of Technology

Safety architecture patterns provide knowledge about large scale design decisions for safety-critical systems. They provide good ways to avoid, detect, and handle faults in software or hardware. In this paper we revise existing architectural safety patterns and organize them to build up a pattern system. We add Goal Structuring Notation diagrams to the patterns to provide a structured overview of their architectural decisions. Based on these diagrams we analyze and present relationships between the patterns. The diagrams can also be used to argue about a systems's safety, which we show with an example.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architecture—*Patterns*; K.4.1 [**Public Policy Issues**] Human Safety; K.6.5 [**Management of computing and information systems**] Security and Protection

ACM Reference Format:
Preschern, C. 2013. Architectural Pattern System for Functional Safety jn 0, 0, Article 0 (0000), 28 pages.

1. INTRODUCTION

Safety-critical systems can directly harm humans or machinery if they malfunction. To ensure that these systems operate properly, they often have to be certified and developed according to safety standards. Safety standards usually provide a big pool of requirements and techniques to achieve system safety. For system architects which are new to the safety domain, it is often difficult to chose which of the provided techniques or which overall system architecture should be used to achieve a safety goal.

To provide safety architects with knowledge about good solutions, we construct a system of architectural safety patterns[1]. We present a structured way how we build up this pattern system from existing safety patterns found in literature. Additionally, we extend these patterns with Goal Structuring Notation (GSN) diagrams, which present the main architectural decisions of the patterns. These diagrams provide a safety architect with a structured approach to argue for the overall system safety. We show with an example how the GSN diagrams can even be used to relate architectural design decisions in the patterns to requirements and techniques of the IEC 61508 safety standard. This approach can as well aid safety architects for arguing for the system safety in particular in the context of safety certification.

This paper is structured as follows: Section 2 gives an overview of existing safety patterns and approaches to build safety pattern systems/languages. Section 3 presents the pattern format we use for our pattern system and Section 4 shows how we bring the patterns into this format by the example of the TRIPLE MODULAR REDUN-

[1] A "pattern system" is similar to a "pattern language", but compared to a pattern language it does not claim to be complete [Buschmann et al., 1996]. Precise definitions about the difference between pattern collections/systems/languages can be found in [Schumacher, 2003]

DANCY pattern. Section 5 presents how our patterns are connected to a pattern system and Section 6 shows the application of the patterns to highlight the benefits of the introduced GSN diagrams. Section 7 concludes this work. In Appendix A, all the patterns of the pattern system are presented, Appendix B shows a collection of safety tactics, and Appendix C shows how we analyzed our patterns to obtain the tactics they use.

2. RELATED WORK

In this section we give an overview of related work that introduces safety patterns (see Table I) and we present related work that collects and structures existing safety patterns.

Table I. Literature which introduces safety-related patterns

Title	**Description**
[Daniels et al., 1997] *"The Reliable Hybrid Pattern - A Generalized Software Fault Tolerant Design Pattern"*	A pattern which includes software fault tolerance techniques (e.g. N-version programming, voting, acceptance test) is presented. The pattern is presented as a generic architecture which explicitly states alternatives in the pattern (e.g. use voting instead of an acceptance test).
[Douglass, 1998] *"Safety-Critical Systems Design"*	The article covers safety architecture patterns and discusses how they can be implemented.
[Saridakis, 2002] *"A System of Patterns for Fault Tolerance"*	This paper introduces several architectural fault-tolerance patterns and discusses how to group them.
[Douglass, 2002] *"Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems"*	Besides other patterns, this book covers safety-related architecture patterns and also includes the patterns from [Douglass, 1998].
[Grunske, 2003] *"Transformational Patterns for the Improvement of Safety Properties in Architectural Specification"*	This paper presents patterns for architecture transformations to increase the overall system safety. Some of the patterns are related to the patterns from [Douglass, 2002].
[Hanmer, 2007] *"Patterns for Fault Tolerant Software"*	The book provides a pattern language of fault-tolerance patterns grouped as error detection, error processing, error mitigation, fault treatment, and architectural patterns.
[Douglass, 2010] *"Design Patterns for Embedded Systems in C"*	The book presents design patterns implemented in C. Some of the presented safety-related patterns come from [Douglass, 2002].
[Armoush, 2010] *"Design Patterns for Safety-critical Embedded Systems"*	This PhD thesis introduces new safety patterns and provides and collects existing safety patterns for embedded systems (mostly [Douglass, 2002] for hardware patterns and software fault tolerance techniques from [Pullum, 2001] brought into pattern notation for software patterns).
[Hampton, 2012] *"Survey of Safety Architectural Patterns"*	This survey presents the application of the patterns from [Armoush, 2010] within a company. Furthermore, some new and rather domain-specific safety patterns are introduced.
[Rauhamäki et al., 2012] *"Architectural Patterns for Functional Safety"*	The paper presents 4 patterns related to separating the safety functionality from non-critical functionality.
[Rauhamäki et al., 2013] *"Patterns for Safety and Control System Cooperation"*	The paper presents 3 safety patterns related to the control systems domain.
[Rauhamäki and Kuikka, 2013] *"Patterns for Controlling System Safety"*	The paper presents 4 safety patterns related to the control systems domain.

[Saridakis, 2002] presents several fault-tolerance patterns in detail and discusses how they can be related to each other. The patterns are classified according to several criteria: pattern complexity, space requirements, time requirements, failure types which are handled by the pattern, and the pattern aim (error detection, recovery, or masking). [Hanmer, 2007] also describes fault-tolerance patterns and presents the patterns and their relationships as a pattern language.

[Armoush, 2010] provides in his PhD thesis a comprehensive collection of safety architecture patterns for embedded systems. Most of the patterns are taken from literature and all are presented in a common pattern format. However, the relationships between the patterns are not described in detail. Armoush provides a tool which lists the patterns and provides detailed information about them (e.g. reliability calculations) when selected.

To bridge the gap between the high-level safety pattern descriptions and their actual implementation, [Gawand et al., 2011] represent safety patterns in UML notation. This is also done by [Sarma et al., 2013] with the pattern catalog of [Armoush, 2010]. This idea was taken further by [Antonino et al., 2012] who introduce a safety-related UML profile to capture architectural safety pattern elements (e.g. voter) and to define rules for them. Based on this idea [Olivera, 2012] implements a repository for safety patterns in UML notation.

The TERESA project applies a model-based approach coupled with a repository of safety/security patterns for embedded systems engineering. A generic metamodel for safety/security patterns is defined which can be used to model domain-specific patterns in the repository [Desnos et al., 2012]. The pattern repository can be accessed with an Eclipse plugin as described on the TERESA project homepage (www.teresa-project.org).

3. APPLIED PATTERN FORMAT

We use the pattern format presented by [Babar, 2007] for all our safety architecture patterns. The pattern format of Babar explicitly provides architectural information with the aim to aid architecture design and evaluation processes. Table II shows which sections this pattern format contains and where we got the information for these sections from. Most of our patterns are based on [Armoush, 2010] and are further elaborated by using other literature on similar safety patterns. For example, Armoush describes the WATCHDOG pattern, which is also described by [Grunske, 2003]. We take Armoush's Watchdog pattern as a starting point and enhance it with information from Grunske. In particular, in this case, we add Grunske's forces, because they are better elaborated.

Table II. Pattern format for our safety architecture patterns

Section	What it contains and where the contents comes from
Pattern Name	The pattern name is taken from the existing pattern - most of which come from [Armoush, 2010].
Pattern Type	Classification into hardware/software and fail-safe/fail-over. This classification comes from the pattern types which we classify during the process of building up the pattern language.
Also Known As	Other names for the pattern used in literature.
Context	The contents of this section comes from existing patterns and was structurally adapted to fit our pattern system.
Problem	The contents of this section comes from existing patterns and was structurally adapted to fit our pattern system.
Forces	The contents of this section comes from existing patterns (mostly from [Grunske, 2003]) and was structurally adapted to fit our pattern system.
Solution	The solution is shortly described in a few sentences and the structure of the safety architecture is shown in a diagram. Most of the diagrams are based on [Armoush, 2010] and [Douglass, 2002].
GSN Diagram	This section contains a Goal Structuring Notation (GSN) diagram which relates the main aim of the pattern to the architectural design decisions which were taken to achieve this aim. GSN is a graphical notation which is often used in the safety domain to describe how a certain goal is achieved. The advantage of using this notation is that it is familiar to safety experts and the resulting pattern GSN diagram can be used to structurally argue about a system's safety. Figure 1 shows the basic elements of GSN and explains them. The GSN diagram is based on information about the usage of basic architectural design decisions (architectural tactics) which are applied in the pattern. We obtain these tactics from pattern descriptions according to a method presented by [Kumar and Prabhakar, 2010b] which we will cover in more detail in the next section.
Consequences	The consequences are split into a part containing general consequences and a part explicitly covering quality-attribute related consequences (e.g. consequences on safety or availability). The information about the consequences mostly comes from the safety patterns from [Armoush, 2010] and [Grunske, 2003].
General Scenarios	This section contains scenarios of the system which can, for example, be used during architecture evaluations. The scenarios are mined from patterns as suggested in [Babar, 2007] by manually searching the problem and solution statements for scenarios for relevant quality attributes (in our case focused on safety). Scenarios are included in the patterns because there are existing safety reasoning frameworks which are based on scenarios ([Wu, 2007]) and the information of the scenarios is also needed to build up our GSN diagrams.
Known Uses	This section presents known uses for the patterns. We added this information by searching for literature which applies the pattern. We just included patterns for which we could find at least three known uses.
Credits	References to previous work on the pattern.

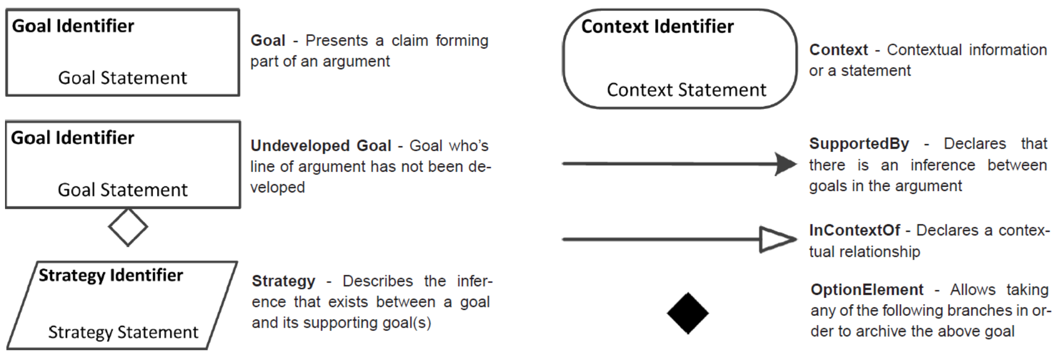

Goal Identifier
Goal Statement
Goal - Presents a claim forming part of an argument
Goal Identifier
Goal Statement
Undeveloped Goal - Goal who's line of argument has not been developed
Strategy Identifier
Strategy Statement
Strategy - Describes the inference that exists between a goal and its supporting goal(s)
Context Identifier
Context Statement
Context - Contextual information or a statement
SupportedBy - Declares that there is an inference between goals in the argument
InContextOf - Declares a contextual relationship
OptionElement - Allows taking any of the following branches in order to archive the above goal

Fig. 1. Explanation of the basic GSN elements

4. SPECIFYING A PATTERN IN THE PROPOSED PATTERN FORMAT

In this section we show how we specify a pattern for our pattern system with the example of the TRIPLE MODULAR REDUNDANCY (TMR) pattern. We focus on how the **GSN Diagram** is built and also describe how we obtain the **General Scenarios** for a pattern.

4.1 Mining Tactics from the Pattern Descriptions

The TMR pattern is mentioned in literature by [Douglass, 2002] and [Armoush, 2010]. We studied both TMR patterns to find text passages which indicate the usage of general safety-related architectural design decisions (safety tactics). We do this as proposed by [Kumar and Prabhakar, 2010a], where architectural tactics are mined from GoF and POSA patterns to find relationships between patterns which use similar tactics. We apply the same method in order to find relationships between safety architecture patterns.

As proposed by [Kumar and Prabhakar, 2010a], we construct a table which includes text passages of the pattern and we give the corresponding tactic that this text passage relates to. For example, the TMR pattern in [Armoush, 2010] says: "*The voter plays a main role in this pattern by applying the voting policy to take the majority from the results which represents the correct actual result.*" This indicates that the pattern applies the *Voting* safety tactic[2].

Table III shows which tactics were mined for the TMR pattern.

4.2 Building the Tactic Topology Model

With the gathered tactics we construct a *Tactic Topology Model* with is also part of the method described by [Kumar and Prabhakar, 2010a]. First, one has to think about the main goal of the pattern (usually found in the patterns' **Intent** section). According to [Kumar and Prabhakar, 2010a], the main tactics which achieve this goal are usually related to the **Intent** or the **Problem** section of the pattern. In the Tactic Topology Model, these main tactics are connected to the patterns' goal with arrows. Further explanation about this connection is given in textual form next to the arrow. The tactics can bring up new goals which have to be achieved by additional tactics - these are also added with arrows and a textual description. In that way, a structured graph containing the patterns' tactics is constructed.

Figure 2 shows the Tactic Topology Model for the TMR pattern. We use the Tactic Topology Models to structurally establish relationships in our pattern system (this is explained in Section 5). Apart from that, the Tactic Topology Models are just intermediate results used to build GSN diagrams and are not included in the patterns.

[2]A list of all safety tactics is available in Appendix B

Table III. Determining the tactics used by the TMR patterns described in [Douglass, 2002] and [Armoush, 2010]

Abstract Section		
Core Intent		**Tactic**
This pattern consists of three identical modules operating in parallel to produce three results that are compared using a voting system to produce a common result		Voting Replication Redundancy
Problem Section		
Problem	**Elaboration of Problem (Scenario)**	**Achieved through Tactic**
How to deal with random faults and single-point of failure in order to increase the safety and reliability of the system without losing the input data in the presence of faults.	The system is fully operational even in case of a single channel failure. A single channel random fault does not lead to a system failure.	Voting
Solution Section		
Solution Description		**Tactic**
The system contains three identical modules or channels operating in parallel. *Test by redundant hardware*		Replication Redundancy
The voter plays a main role in this pattern by applying the voting policy to take the majority from the results which represents the correct actual result. *Fault detection and diagnosis (Voting)*		Voting
Consequences Section		
Consequence Description		**Tactic**
This pattern has a high recurring cost due to the using of three parallel modules. So, the recurring cost is 300% comparing to the basic system.		Replication Redundancy
The cost of voter which is normally a simple hardware circuit that depends on the type of the output control signal and the implementation method.		Voting
Implementation Section		
Implementation Description		**Tactic**
To implement this pattern, the designer should replicate the channel which includes the replication of the hardware as well as software.		Replication Redundancy

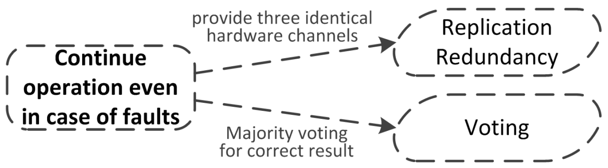

provide three identical hardware channels
Replication Redundancy
Continue operation even in case of faults
Majority voting for correct result
Voting

Fig. 2. Tactic Topology Model for the TMR pattern

4.3 Building the Goal Structuring Notation Diagram

Based on the Tactic Topology Model, we construct the GSN diagrams for the patterns. The GSN diagrams contain the tactics from the Tactic Topology Model and they additionally contain general scenarios which are mined from the pattern descriptions. This scenario mining is done as proposed in [Babar, 2007] by searching the problem and solution statements for safety-related scenarios. The scenarios found for the TMR pattern are shown in Table III under *Problem Section*. All our GSNs start with the main goal to maintain system safety. This main goal is split up into subgoals with the scenarios which we obtained from the patterns. If the scenarios are independent from each other, then they are put on the same level in the GSN. If a scenario depends on another scenario (as it is the case for the TMR pattern), then it is modeled as a subgoal of the scenario it depends on. The tactics which are necessary to achieve a GSN goal are put below this (sub-)goal as a GSN strategy which has the title of the tactic and which contains additional information (taken from the textual description of the Tactic Topology Model arrow connections) as GSN strategy description. GSN context elements are added to the GSN diagram if information of the pattern's context section is relevant for the GSN goals.

Figure 3 shows the GSN diagram of the TMR pattern. We can see that it consists of the tactics of the TMR pattern Tactic Topology Model from Figure 2 and of the scenarios of the TMR pattern from Table III. The complete TMR pattern including the here constructed GSN diagram is shown in Appendix A on page 12.

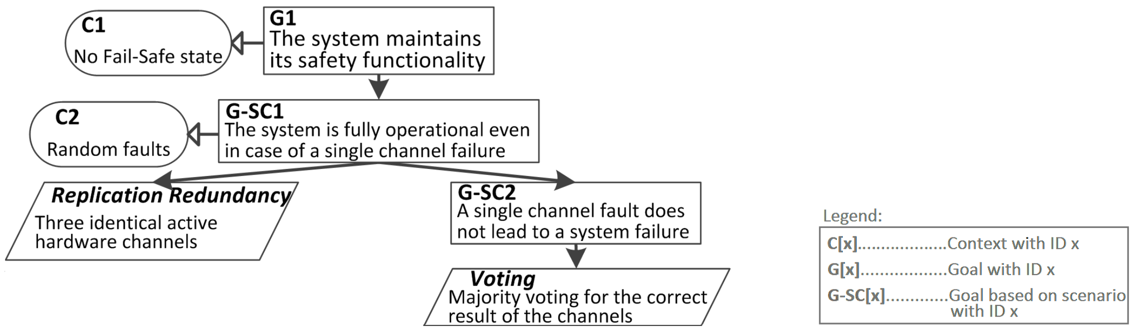

C1
No Fail-Safe state
G1
The system maintains its safety functionality
C2
Random faults
G-SC1
The system is fully operational even in case of a single channel failure
Replication Redundancy
Three identical active hardware channels
G-SC2
A single channel fault does not lead to a system failure
Voting
Majority voting for the correct result of the channels
Legend:
C[x]..................Context with ID x
G[x].................Goal with ID x
G-SC[x]............Goal based on scenario with ID x

Fig. 3. GSN diagram of the TMR pattern

5. ORGANIZING SAFETY PATTERNS TO A PATTERN SYSTEM

To obtain the relationships between the patterns, we use the approach presented by [Kumar and Prabhakar, 2010b]. They compare Tactic Topology Models of patterns and define a mapping between Tactic Topology Model predicates and pattern relationships. For example, if the Tactic Topology Models of two different patterns are equal, then Kumar and Prabhakar say that these patterns are similar. Table IV shows all kinds of relationships defined by Kumar and Prabhakar. To find all relationships in a pattern system, every patterns' Tactic Topology Model has to be compared to the Tactic Topology Models of all other patterns and every such Tactic Topology Model pair has to be checked for all the predicates described in Table IV.

Table IV. Description of pattern relationships (slightly modified from [Kumar and Prabhakar, 2010a])

Relationship	**Description**	**Tactic Topology Model predicate**
is an alternative	Patterns A and B solve the same problem, but propose different solutions.	$SourceNode(A) = SourceNode(B)$ **AND** $Graph(A) \neq Graph(B)$
uses	A sub-problem of pattern A is similar to the problem addressed by pattern B.	$Graph(A) \supset Graph(B)$
refines	Pattern B provides a more detailed solution than pattern A.	$SourceNode(A) = SourceNode(B)$ **AND** $Graph(A) \subset Graph(B)$
specializes	The solution of pattern B is a special case of the solution of pattern A. *Example: Pattern B specializes pattern A if they have the same graph structure, but pattern B uses a refined tactic where pattern A uses a more general tactic (e.g. B uses Replication Redundancy where A uses Redundancy).*	$Graph(A) \subset generalizedGraph(B)$
is similar	Patterns A and B provide the same solution to a similar problem *Example: Pattern B is similar to pattern A if they have the same graph structure and they use two related refined tactics. E.g. A uses Replication Redundancy and B uses Diverse Redundancy*	$generalizedGraph(A) \equiv generalizedGraph(B)$

We applied this approach to our safety patterns to structurally build the relationships in our pattern system. We built the Tactic Topology Models as described in Section 4 for all our patterns. Then we compared each Tactic Topology Model with one another and checked for the predicates defined in Table IV. This delivers us the relationships between all the patterns for our pattern system.

Figure 4 shows our safety patterns and their relationships which we obtained with the described approach. We can see that the approach to find pattern relationships worked out pretty well for our safety patterns. All the relationships between the patterns seem to be comprehensible. For example, according to the relationships obtained through the Tactic Topology Model comparison, the TRIPLE MODULAR REDUNDANCY pattern is a specialization of the M-OUT-OF-N pattern and is similar to the N-VERSION PROGRAMMING pattern which is both reasonable.

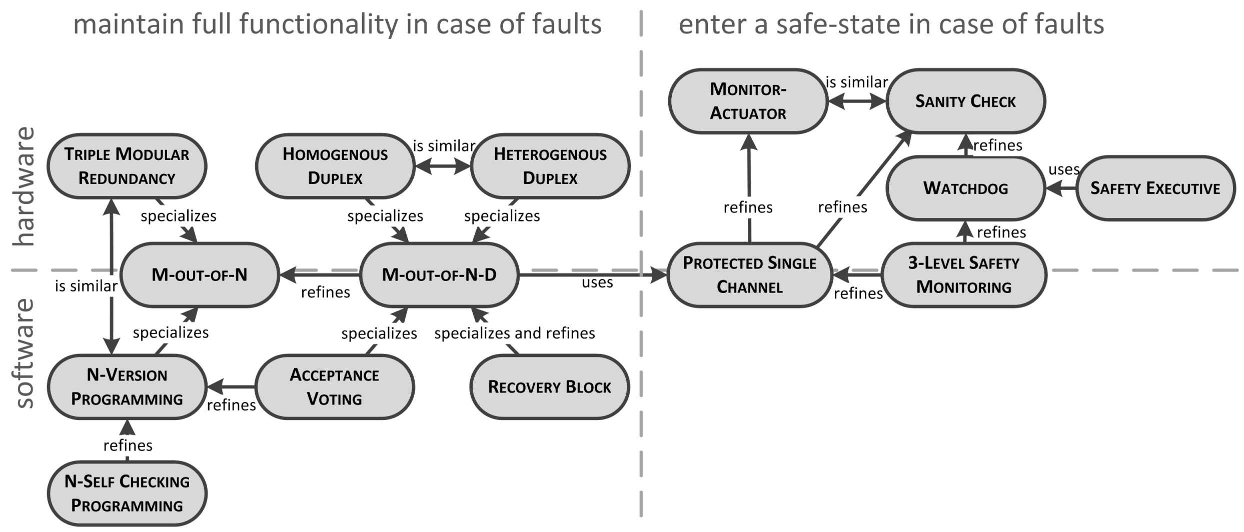


Fig. 4. Safety Architecture Pattern System

To not overload the the pattern-relationship representation, we did not explicitly annotate the *is alternative* relationships, but instead grouped patterns which are alternatives to one another into the group of patterns trying to maintain a safe-state in case of faults and the group of patterns providing full system functionality in case of faults. Additionally, we divided the patterns into software and hardware patterns as already suggested by [Armoush, 2010]. However, the classification of software and hardware patterns is not very strict. Some of the patterns are intended for either software or hardware, but could also be implemented for the other. For example, the WATCHDOG pattern is a hardware pattern, but could also be realized in software by a timer which watches the execution of another program.

The patterns in our safety pattern system are mostly taken from [Armoush, 2010], because these patterns already provide a good collection of other patterns in literature and they focus on rather large-scale architectural design decisions which is the main focus of our pattern system. We included all but one of Armoush's patterns. We excluded one pattern (RECOVERY BLOCK WITH BACKUP VOTING), because we could not find any known uses for it. Additionally to Armoush's patterns we included the M-OUT-OF-N and the M-OUT-OF-N-D pattern, which are based on architectures described in the IEC 61508 safety standard. For each of the patterns from Figure 4, we present the full pattern in Appendix A. Additionally, we provide the tables which show how we related the architectural tactics to the safety patterns as well as the Tactic Topology Models in Appendix C.

6. APPLYING THE TMR PATTERN TO AN EXAMPLE

In this section we show how one of the safety patterns can be applied to an example system. We describe a system found in literature and show which additional benefits could be gained if the architect used our patterns.

The system described in [Alvarez et al., 2005] is a Programmable Logic Device (PLD) safety architecture to be used in control system applications. The basic system (which does not yet fulfill the system safety requirements) consists of a component to handle sensor values (Safe Input), a processing unit which computes output values (CPU), and an interface element for actuators (Safe Output). The basic system is shown in Figure 5.

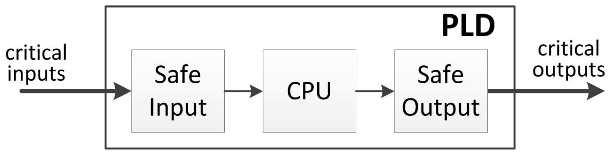


Fig. 5. Basic PLD control system architecture

The proposed safety architecture described in [Alvarez et al., 2005] applies (but not explicitly mentions) the M-OUT-OF-N-D PATTERN (the full pattern is presented in Appendix A - page 12). The architecture uses three identical redundant versions of the basic system architecture and the correct output of these three channels is decided by a majority voter. The three channels are diagnosed with self-tests and if the diagnosis fails, the corresponding channel informs the voter that it does not function properly. The voter then excludes this channel from the vote. The overall safety architecture is shown in Figure 6.

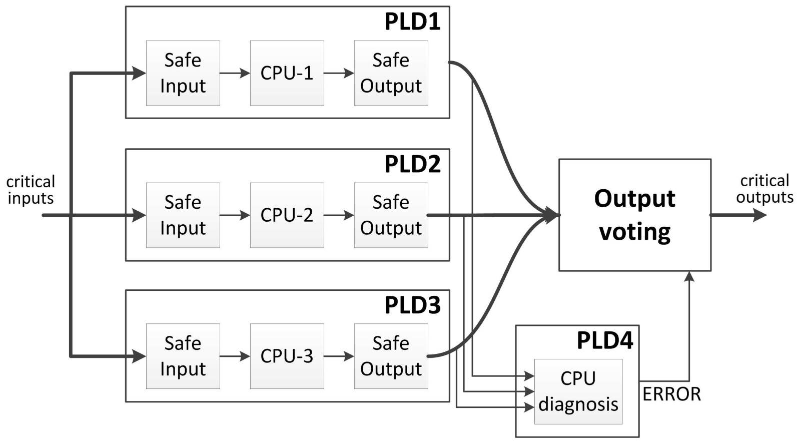


Fig. 6. Safe PLD control system architecture [Alvarez et al., 2005]

In the GSN diagram of the M-OUT-OF-N-D PATTERN, for an architecture several decisions have to be made. For example, we can see that the pattern ether uses the *Replication Redundancy* or the *Diverse Redundancy* tactic. The presented architecture uses *Replication Redundancy* (identical hardware channels), therefore we just consider this redundancy tactic and omit *Diverse Redundancy* from the GSN. This already shows how the GSN diagrams can flexibly be used to describe alternatives for a pattern. Furthermore, the architecture uses *Condition Monitoring* (checks if CPU outputs relate to a reference value) and *Voting* (majority voting). Both of these tactics were also chosen from the set of tactic options presented in the patterns' GSN diagram. Table V lists all the tactics that the architecture uses and presents the IEC 61508 methods which are related to these tactics (taken from

Appendix B). With this table, a safety architect gets a quick overview of methods presented in the safety standard which are relevant for the specific system architecture.

Table V. IEC 61508 methods suitable for the M-OUT-OF-N-D PATTERN

Tactic	**IEC 61508 method**
Replication Redundancy	A.2.1 Tests by redundant hardware A.2.5 Monitored redundancy A.3.5 Reciprocal comparison by software A.4.5 Block replication A.6.3 Multi-channel output A.6.5 Input comparison/voting A.7.3 Complete hardware redundancy A.7.5 Transmission redundancy
Condition Monitoring	A.1.1 Failure detection by online monitoring A.6.4 Monitored outputs A.9 Temporal and logical program monitoring A.13.1 Monitoring
Voting	A.1.4 Majority voter

From the list of IEC 61508 methods, a safety architect can now choose methods which are appropriate for the specific system. For the the specific design decisions taken in [Alvarez et al., 2005], the IEC 61508 methods that are eligible and are actually used are the following:

—A.1.1 Failure detection by online monitoring
—A.1.4 Majority voter
—A.2.1 Tests by redundant hardware
—A.6.4 Monitored output
—A.13.1 Monitoring

In the GSN diagram of the M-OUT-OF-N-D PATTERN, the tactics can now be replaced with the methods that are actually used in the architecture. Figure 7 shows the resulting GSN diagram which can be used by safety architects to reason about the overall system safety by structurally referring to methods suggested by the safety standard. This gives a structured connection between the goal to maintain the overall system safety down to the actually applied methods. Such a connection can be used during the system certification to argue how the safety goals are achieved by a specific system architecture.

Reasoning about the safety of a system by constructing GSN diagrams based on scenarios was already suggested in [Wu, 2007], where the argument is made that a system which covers all its goals mentioned in relevant scenarios is reasonably safe.

The safety standard describes in detail how to implement the methods which are now present in the GSN. Table VI shows additional information about the applied safety methods taken from the IEC 61508 safety standard. With this information the system architect gets guidance of how to realize the safety methods. Now, the safety architect just has to think about the remaining undeveloped goals (G2, G5, G6 in Figure 7 of the GSN diagram to obtain a complete safety argumentation for the architecture.

We saw, that when applying the suggested safety patterns, additionally to the solution description and the described consequences, a safety architect gets:

—A GSN diagram for the architecture which can be taken as a starting point to develop a structured argument about the system's safety.

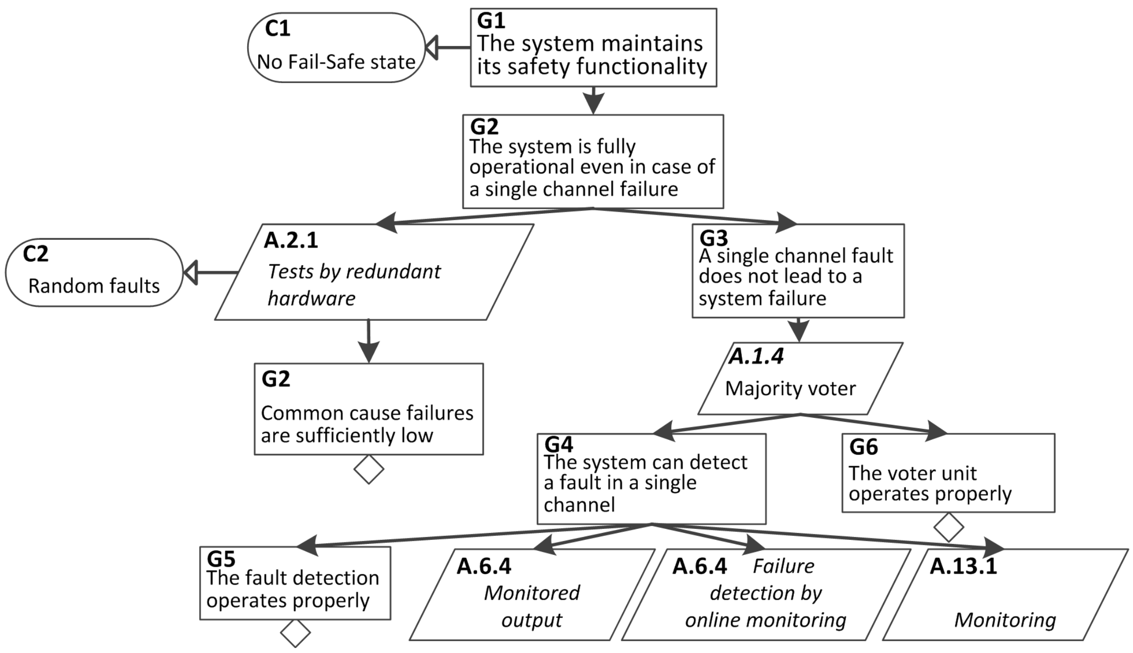

C1
No Fail-Safe state
G1
The system maintains its safety functionality
G2
The system is fully operational even in case of a single channel failure
A.2.1
Tests by redundant hardware
C2
Random faults
G3
A single channel fault does not lead to a system failure
G2
Common cause failures are sufficiently low
A.1.4
Majority voter
G4
The system can detect a fault in a single channel
G6
The voter unit operates properly
G5
The fault detection operates properly
A.6.4
Monitored output
A.6.4 Failure detection by online monitoring
A.13.1
Monitoring

Fig. 7. GSN for the safe PLD control system architecture

Table VI. Safety methods used for the PLD control system architecture (taken form IEC 61508)

Method	Aim	Description
Failure detection by online monitoring	To detect failures by monitoring the behaviour of the E/E/PE safety-related system in response to the normal (on-line) operation of the equipment under control (EUC).	Under certain conditions, failures can be detected using information about (for example) the time behaviour of the EUC. For example, if a switch, which is part of the E/E/PE safety-related system, is normally actuated by the EUC, then if the switch does not change state at the expected time, a failure will have been detected. It is not usually possible to localise the failure.
Majority Voter	To detect and mask failures in one of at least three hardware channels.	A voting unit using the majority principle (2 out of 3, 3 out of 3, or m out of n) is used to detect and mask failures. The voter may itself be externally tested, or it may use selfmonitoring technology.
Tests by redundant hardware	To detect failures using hardware redundancy, i.e. using additional hardware not required to implement the process functions.	Redundant hardware can be used to test at an appropriate frequency the specified safety functions.
Monitored output	To detect individual failures, failures caused by external influences, timing failures, addressing failures, drift failures (for analogue signals) and transient failures	This is a dataflow-dependent comparison of outputs with independent inputs to ensure compliance with a defined tolerance range (time, value). A detected failure cannot always be related to the defective output. This measure is only effective if the dataflow changes during the diagnostic test interval.
Monitoring	To detect the incorrect operation of an actuator.	The operation of the actuator is monitored. The redundancy introduced by this monitoring can be used to trigger emergency action.

—A list of IEC 61508 methods which are related to the overall architecture. A safety architect gets a pool of methods which could be relevant for the chosen architecture. Furthermore, the standard provides additional information about how to implement the methods.

—A connection between the safety goals of the overall architecture and IEC 61508 methods which fulfill these goals. This allows a safety architect to structurally present a safety certification authority how the applied methods which are suggested by the standard are combined to achieve a safe system.

7. CONCLUSION

We presented a system of safety patterns and described their relationships to each other. The patterns include a GSN diagram for safety reasoning.

This pattern system allows safety engineers to easily get an overview of commonly used system architectures and their safety-related consequences. Additionally, when using a pattern, the safety engineer can construct a GSN diagram for his architecture based on the GSN diagrams in the patterns. The GSN representation for the patterns is very suitable, because many of the patterns have alternatives which just differ in changing a single design decision. For example, each of the patterns addressing random faults by using *Replication Redundancy* can easily be used to handle systematic faults as well if *Diverse Redundancy* is used instead. With the GSN representation such alternatives can easily be modeled by simply exchanging a tactic of the pattern (see the M-OUT-OF-N-D PATTERN for example). Similarly, variants of a pattern can be modeled by refining the pattern by an additional tactic. The systematic GSN notation allows to easily integrate additional safety patterns into our pattern system and it allows to reason about safety-specific consequences of these patterns by having a look at the consequences of the added tactic.

We think that the presented system for architectural safety patterns provides safety engineers a good overview of safety architectures and it allows to connect IEC 61508 methods to high level architectures. This is particularly important during safety certification and offers safety engineers a new way to argue about how their architecture achieves safety goals.

ACKNOWLEDGMENTS

We would like to thank our shepherd Brahim Hamid who gave us valuable feedback on this paper. He provided us good improvement suggestion to make this paper easier to read.

REFERENCES

ALVAREZ, Jacobo et al. (2005). Safe PLD-based programmable controllers. In: *International Conference on Field Programmable Logic and Applications*. IEEE, 559–562.

ANTONINO, Pablo Oliveira, Thorsten KEULER, and Pablo ANTONINO (2012). Towards an Approach to Represent Safety Patterns. In: *The Seventh International Conference on Software Engineering Advances (ICSEA)*. c, 228–237.

ARMOUSH, Ashraf (2010). Design patterns for safety-critical embedded systems. PhD thesis. RWTH Aachen University.

BABAR, M.A. (2007). Improving the Reuse of Pattern-Based Knowledge in Software Architecting. In: *EuroPLoP*. Lero, Ireland, 7–11.

BUSCHMANN, Frank et al. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons.

DANIELS, Fonda, Kalhee KIM, and Mladen A VOUK (1997). The Reliable Hybrid Pattern A Generalized Software Fault Tolerant Design Pattern. In: *European Conference on Pattern Language of Programs (EuroPLoP)*, 1–9.

DESNOS, Nicolas et al. (2012). Towards a Security and Dependability Pattern Development Technique for Resource Constrained Embedded Systems. In: *4th International Conference on Software Quality, Process Automation in Software Development*. Springer, Vienna, Austria, 193–204.

DOUGLASS, Bruce Powel (1998). Safety-Critical Systems Design. *Electronic Engineering* 70, 862.

DOUGLASS, Bruce Powel (2002). *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Pearson.

DOUGLASS, Bruce Powel (2010). *Design Patterns for Embedded Systems in C*. Elsevier.

GAWAND, Hemangi, RS MUNDADA, and P. SWAMINATHAN (2011). Design Patterns to Implement Safety and Fault Tolerance. *International Journal of Computer Applications* 18, 2, 6–13.

GRUNSKE, Lars (2003). Transformational Patterns for the Improvement of Safety Properties in Architectural Specification. In: *Nordic Conference on Pattern Languages of Programs (VikingPLoP)*.

HAMPTON, Paul (2012). Survey of safety Architectural Patterns. In: *Achieving Systems Safety*. February 2012. Springer London, London, 7–9.

HANMER, Robert S. (2007). *Patterns for Fault Tolerant Software*. Wiley.

KUMAR, Kiran and T.V. PRABHAKAR (2010a). Design Decision Topology Model for Pattern Relationship Analysis. In: *1st Asian Conference on Pattern Languages of Programs (AsianPLoP 2010)*.

KUMAR, Kiran and T.V. PRABHAKAR (2010b). Pattern-oriented Knowledge Model for Architecture Design. In: *17th Conference on Pattern Languages of Programs (PLoP)*.

OLIVERA, Andre Rodrigues (2012). Taim : A Safety Pattern Repository, BsC thesis. Federal University of Rio Grande do sul.

PULLUM, L. (2001). *Software fault tolerance techniques and implementation*. Artech House.

RAUHAMÄKI, Jari and Seppo KUIKKA (2013). Patterns for control system safety. In: *18th European Conference on Pattern Languages of Programs (EuroPLoP)*.

RAUHAMÄKI, Jari, Timo VEPSÄLÄINEN, and Seppo KUIKKA (2012). Architectural patterns for functional safety. In: *Nordic Conference on Pattern Languages of Programs (VikingPLoP)*.

RAUHAMÄKI, Jari, Timo VEPSÄLÄINEN, and Seppo KUIKKA (2013). Patterns for safety and control system cooperation. In: *Nordic Conference on Pattern Languages of Programs (VikingPLoP)*.

SARIDAKIS, Titos (2002). A System of Patterns for Fault Tolerance. In: *EuroPLoP*.

SARMA, U V R, Sahith RAMPELLI, and P PREMCHAND (2013). A Catalog of Architectural Design Patterns for Safety-Critical Real-Time Systems. *International Journal of Engineering Research and Applications* 3, 1, 125–131.

SCHUMACHER, Markus (2003). *Security Engineering with Patterns*. Springer.

WU, Weihang (2007). Architectural Reasoning for Safety- Critical Software Applications. PhD thesis. University of York.

A. SAFETY ARCHITECTURE PATTERNS

This publication has been shortened and this section has been skipped, because it contains the same information as present in Appendix A in Publication 4: "Safety Architecture Pattern System with Security Aspects". The full publication of the current paper including this section is available at the ACM Digital Library.

B. SAFETY TACTICS

This publication has been shortened and this section has been skipped, because it contains the same information as present in Appendix B in Publication 4: "Safety Architecture Pattern System with Security Aspects". The full publication of the current paper including this section is available at the ACM Digital Library.

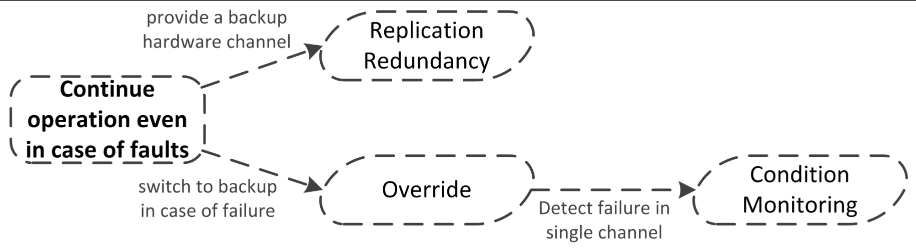
C. RETRIEVING TACTICS FROM SAFETY PATTERNS

In this section we present the tactic mining process as proposed in Section 4 and its results. We analyzed the safety patterns regarding the tactics they use as described by [Kumar and Prabhakar, 2010] where all pattern sections are analyzed and compared to tactic descriptions. If a part of a pattern section is similar to a safety tactic, then this tactic is used in the pattern. An overview of the considered safety tactics is given in Appendix B.

In the left column of the following tables we give the original statements from the context, problem, solution, consequences, and implementation sections of the patterns. In the right column we give the tactic which is addressed by the description on the left. Additionally we elaborate the problem section as described by [Babar, 2007] to gather general scenarios for the patterns. For tactic mining we put special focus on the solution section as suggested by [Zhu et al., 2004].

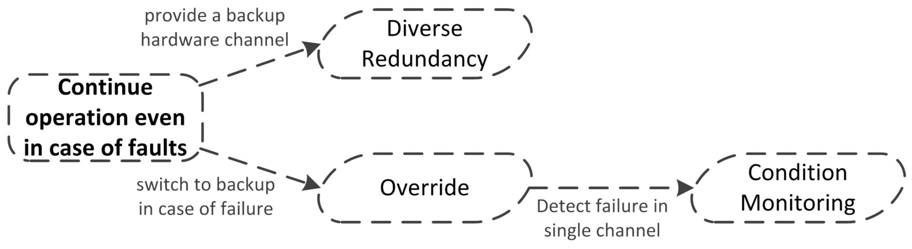
With the gathered tactics we construct a *Tactic Topology Model* which was introduced by [Kumar and Prabhakar, 2010]. The model takes the main tactic to handle the pattern problem as root node. If the application of this tactic introduces new goals, then additional tactics to handle them are added as child nodes. The edges of the model give further explanation of the tactic application.

HOMOGENOUS DUPLEX PATTERN

Abstract Section		
Core Intent		**Tactic**
It is a hardware pattern that is used to increase the safety and reliability of the system by providing a replication of the same module (Modular redundancy) to deal with the random faults.		Replication Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
Make the system continue operating in the presence of a fault	The system is fully operational even in case of a single channel failure. A single channel random fault does not lead to a system failure. The system can detect a fault in a single channel.	Replication Redundancy Override
Solution Section		
Solution Description		**Tactic**
The system consists of two identical modules; a primary (active) module and secondary (standby) *Test by redundant hardware*		Replication Redundancy
There is a fault detection unit that monitors the primary module and switches to the secondary module when a fault appears in the primary. *Fault detection and diagnosis (Comparator and Acceptance Test)*		Override
This method performs a check on the two channels by checking for input valid data within a given range and by checking the output signals from the two modules whether they are valid or not.		Condition Monitoring
Consequences Section		
Consequence Description		**Tactic**
When a fault is detected in the primary channel, the switch circuit switches over to the secondary channel		Override
Implementation Section		
Implementation Description		**Tactic**
To implement this pattern, the computational channel should be duplicated		Replication Redundancy
Tactic Topology Model		
		

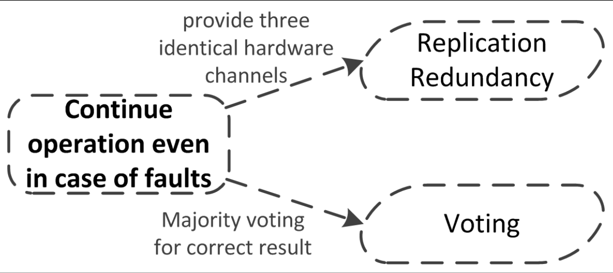
Continue operation even in case of faults
provide a backup hardware channel
Replication Redundancy
switch to backup in case of failure
Override
Detect failure in single channel
Condition Monitoring

Heterogenous Duplex Pattern

Abstract Section		
Core Intent		**Tactic**
Solution for embedded system with no fail-safe-state in a situation that includes high random failure and high systematic failure rate.		Diverse Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to deal with systematic faults as well as random faults in order to increase the safety and reliability of the system. How to make the system continue operating in the presence of a fault in one of the system components	The system is fully operational even in case of a single channel failure. A single channel random/systematic fault does not lead to a system failure. The system can detect a fault in a single channel.	Diverse Redundancy Override
Solution Section		
Solution Description		**Tactic**
The system consists of two modules (channels) with the same functionality; a primary (active) module and secondary (standby). *Test by redundant hardware*		Redundancy
There is a fault detection unit that monitors the primary module and switches to the secondary module when a fault appears in the primary. *Fault detection and diagnosis (Comparator and Acceptance Test)*		Condition Monitoring
The two modules have independent designs or implementation methods, which gives this pattern the ability to handle systematic faults as well as random faults. *Diverse Hardware*		Diverse Redundancy
When there is a fault in the primary channel, the comparator has to detect and to identify the faulty channel, then it generates an instruction to the switch circuit to switch to the secondary channel		Condition Monitoring Override
This method performs a check on the two channels by checking for input valid data within a given range and by checking the output signals from the two modules whether they are valid or not.		Condition Monitoring
Consequences Section		
Consequence Description		**Tactic**
This pattern includes two independent and diverse modules		Diverse Redundancy
When a fault is detected, the switch circuit switches over to the secondary channel		Redundancy Override
Implementation Section		
Implementation Description		**Tactic**
To implement this pattern, the computational channel should be duplicated		Redundancy
The duplicated modules should be implemented using independent designs or independent methods to avoid common systematic faults. It is more preferable to use different software versions that are designed by different teams and using different algorithms, when it is possible.		Diverse Redundancy
Tactic Topology Model		
		

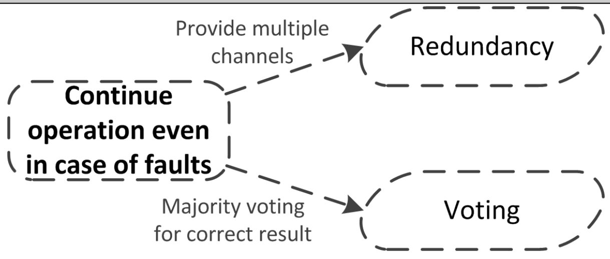
provide a backup hardware channel
Diverse Redundancy
Continue operation even in case of faults
switch to backup in case of failure
Override
Detect failure in single channel
Condition Monitoring

TRIPLE MODULAR REDUNDANCY PATTERN

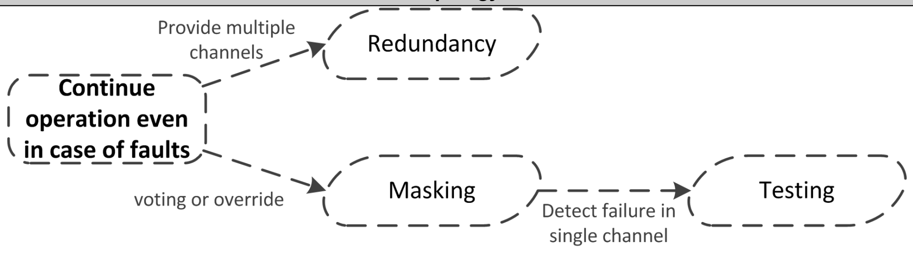
Abstract Section		
Core Intent		**Tactic**
This pattern consists of three identical modules operating in parallel to produce three results that are compared using a voting system to produce a common result		Voting Replication Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to deal with random faults and single-point of failure in order to increase the safety and reliability of the system without losing the input data in the presence of faults.	The system is fully operational even in case of a single channel failure. A single channel random fault does not lead to a system failure.	Voting
Solution Section		
Solution Description		**Tactic**
The system contains three identical modules or channels operating in parallel. *Test by redundant hardware*		Replication Redundancy
The voter plays a main role in this pattern by applying the voting policy to take the majority from the results which represents the correct actual result. *Fault detection and diagnosis (Voting)*		Voting
Consequences Section		
Consequence Description		**Tactic**
This pattern has a high recurring cost due to the using of three parallel modules. So, the recurring cost is 300% comparing to the basic system.		Replication Redundancy
The cost of voter which is normally a simple hardware circuit that depends on the type of the output control signal and the implementation method.		Voting
Implementation Section		
Implementation Description		**Tactic**
To implement this pattern, the designer should replicate the channel which includes the replication of the hardware as well as software.		Replication Redundancy
Tactic Topology Model		
		

provide three identical hardware channels
Replication Redundancy
Continue operation even in case of faults
Majority voting for correct result
Voting

M-out-of-N Pattern

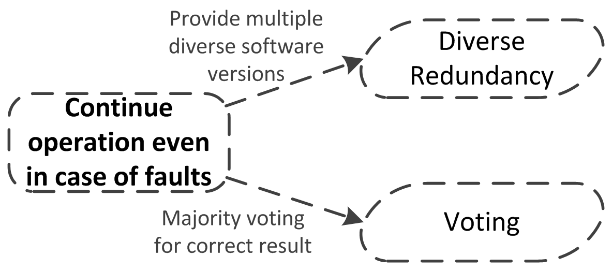
Abstract Section		
Core Intent		**Tactic**
The M-oo-N redundancy requires that at least M components succeed out of the total N parallel modules for the system to succeed.		Voting Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to deal with random or systematic faults in order to increase the safety and reliability of the system without losing the input data.	The system is fully operational even in case of a single channel failure. A single channel random/systematic fault does not lead to a system failure.	Voting
Solution Section		
Solution Description		**Tactic**
The pattern structure contains N identical modules or channels *Test by redundant hardware*		Redundancy
The voting element plays the main role in this pattern since it is used to find the possible correct result by performing the M-oo-N voting strategy *Fault detection and diagnosis (Voting)*		Voting
Consequences Section		
Consequence Description		**Tactic**
This pattern has little influence on the executing time, since the N modules are running separately.		Redundancy
Implementation Section		
Implementation Description		**Tactic**
For homogeneous implementation of this pattern, the designer should use the same hardware as well as the software for all the channels.		Replication Redundancy
If the hardware diversity concept is used in the implementation to solve the problem of systematic faults, then the possible deviation in value or time between the correct outputs should be taken into consideration in the design of the voting system		Diverse Redundancy
Tactic Topology Model		
 Continue operation even in case of faults Provide multiple channels Redundancy Majority voting for correct result Voting		

M-OUT-OF-N-D PATTERN

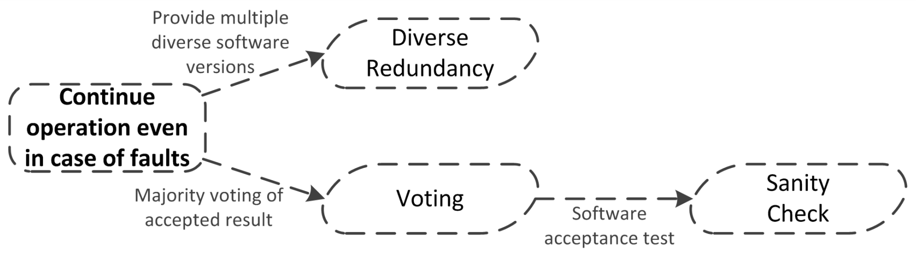
Papers about the MooND Architecture	
Citation	**Tactic**
In addition, if the diagnostic tests in either channel detect a fault then the output voting is adapted so that the overall output state then follows that given by the other channel [International Electrotechnical Commission, 2010]	Checking Override Voting
MooND means M out of N channel architecture with diagnostic [H. Yang and X. Yang, 2010]	Replication Redundancy Diverse Redundancy Voting Checking
The architecture consists of equipment with inputs and outputs wired in parallel [Goble, 1998]	Redundancy
Faulty sensors and actuators can also be detected and isolated from the function with proper diagnostics. This method is designated with an abbreviation MooND, where 'D' stands for diagnostics. [Varjoranta, 2012]	Checking
Tactic Topology Model	
 Continue operation even in case of faults Provide multiple channels Redundancy voting or override Masking Detect failure in single channel Testing	

N-Version Programming Pattern

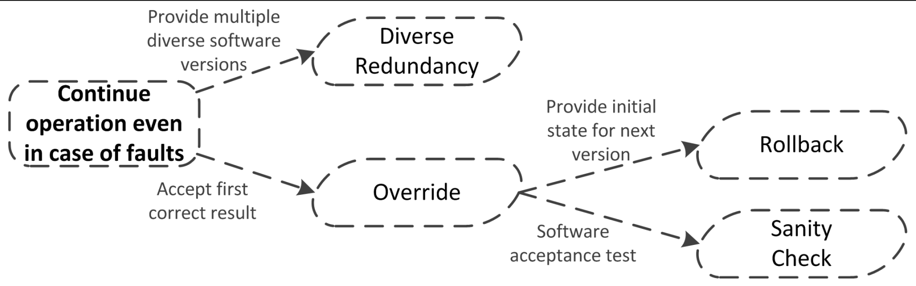
Abstract Section		
Core Intent		**Tactic**
independent generation of N>=2 functionally equivalent software modules called 'versions' from the same initial specification		Diverse Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
Overcome software faults, which may remain after the software development.	The system is fully operational even in case of a failure of a software version. A software fault in a single version does not lead to a system failure	Diverse Redundancy Voting
Solution Section		
Solution Description		**Tactic**
The N-Version Programming Pattern is based on the concept of independent generation of functionally equivalent N versions from the same initial specification. *Diverse programming*		Diverse Redundancy
The outputs of these versions are sent to the voter which executes a voting strategy to determine the best correct output. *Fault detection with voting*		Voting
Consequences Section		
Consequence Description		**Tactic**
The main drawbacks of the NVP Pattern are the complexity of developing independent N-versions		Diverse Redundancy
Implementation Section		
Implementation Description		**Tactic**
The success of the NVP Pattern depends on the independent development of the required N versions and the level of diversity in these versions to avoid the common failures		Diverse Redundancy
There are several voting techniques that can be used in this pattern to implement the voter component		Voting
Tactic Topology Model		


Provide multiple diverse software versions
Diverse Redundancy
Continue operation even in case of faults
Majority voting for correct result
Voting

ACCEPTANCE VOTING PATTERN

Abstract Section		
Core Intent		**Tactic**
Acceptance Voting Pattern is based on the independent generation of N>=2 functionally equivalent software modules called ŞversionsŤ from the same initial specification		Diverse Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to overcome the software faults, which may remain after the software development, in order to improve the software reliability and safety	The system is fully operational even in case of a failure of a software version. A software fault in a single version does not lead to a system failure. A fault in a single software version is detected.	Diverse Redundancy Voting
Solution Section		
Solution Description		**Tactic**
It includes N independent and functionally equivalent versions that are typically executed in parallel to perform the required task. *Diverse programming*		Diverse Redundancy
The output of each version is tested for correctness using an acceptance test.		Sanity Check
Those results that pass the acceptance test are then used by the voting algorithm to generate the final result.		Voting
Fault detection and diagnosis (Voting and Acceptance Test)		Sanity Check Voting
Consequences Section		
Consequence Description		**Tactic**
The voter has to wait for the outputs of all versions to be checked by the acceptance test before applying the voting algorithm		Voting Sanity Check
The development cost include the development of independent and functionally equivalent N versions.		Diverse Redundancy
-		-
Implementation Section		
Implementation Description		**Tactic**
The quality of the acceptance. Thus, it should be carefully designed to detected most of the possible software faults.		Sanity Check
The independent development of the required N versions and the level of diversity in these versions to avoid the common failures		Diverse Redundancy
The use of a suitable voting technique		Voting
Tactic Topology Model		
 Provide multiple diverse software versions Diverse Redundancy Continue operation even in case of faults Majority voting of accepted result Voting Software acceptance test Sanity Check		

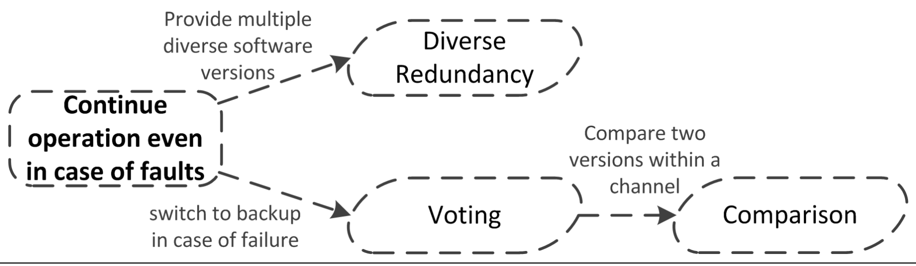
RECOVERY BLOCK PATTERN

Abstract Section		
Core Intent		**Tactic**
It includes N diverse, independent, and functionally equivalent software modules called 'versions'		Diverse Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to overcome the software faults, which may remain after the software development, in order to improve the software reliability and safety	The system is fully operational even in case of a failure of a software version. A software fault in a single version does not lead to a system failure. A fault in a single software version is detected.	Diverse Redundancy Override
Solution Section		
Solution Description		**Tactic**
After the execution of the primary version, the acceptance test is executed to check if the outcome is reasonable and to detect any possible erroneous result. *Fault detection and diagnosis*		Sanity Check
The system state should be restored to its original state and an alternate version will be invoked to repeat the same computations. *Recovery block*		Rollback
An overall system failure is reported to execute the available safety action such as switching the system into its fail-safe sate, or to shutdown the system.		Override
Diverse programming		Diverse Redundancy
The primary alternate is the one which is intended to be used normally to perform the desired operation		Override
Consequences Section		
Consequence Description		**Tactic**
The normal recovery block runs the independent ver- sions serially on a single hardware unit.		Diverse Redundancy
The main drawbacks of the RB are the high dependency on the quality of the acceptance test.		Sanity Check
Implementation Section		
Implementation Description		**Tactic**
The acceptance test should be carefully designed to detected most of the possible software faults		Sanity Check
The independent development of the required N versions and the level of diversity in these versions to avoid the common failures.		Diverse Redundancy
Tactic Topology Model		
		

Provide multiple diverse software versions
Diverse Redundancy
Continue operation even in case of faults
Provide initial state for next version
Rollback
Accept first correct result
Override
Software acceptance test
Sanity Check

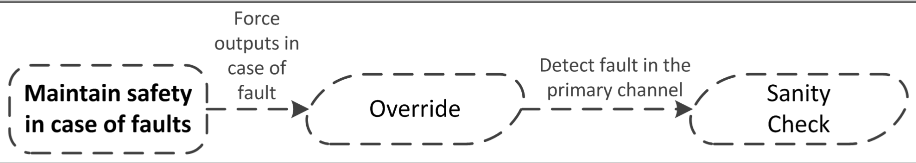
N-SELF CHECKING PROGRAMMING PATTERN

Abstract Section		
Core Intent		**Tactic**
This pattern includes an independent generation of N>=4 functionally equivalent software modules called 'versions' from the same initial specification		Diverse Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to overcome the software faults, which may remain after the software development, in order to improve the software reliability and safety	The system is fully operational even in case of a failure of a software version. A software fault in a single version does not lead to a system failure.	Diverse Redundancy Voting
Solution Section		
Solution Description		**Tactic**
NSCP uses software design diversity and error detection by self-checking programming diverse programming. *Diverse programming*		Diverse Redundancy
each component includes two independent and functionally equivalent versions that run in parallel and are self checked using a comparison algorithm. *Fault detection and diagnosis with a comparator*		Diverse Redundancy Voting
When the running component fails due to different results from its versions, a spare component is invoked to start delivering the required functionality		Diverse Redundancy
If there is no agreement between the two versions, then this component is discarded and a signal is generated to indicate a fault in this component and the selector switches to the next spare component.		Comparison
Consequences Section		
Consequence Description		**Tactic**
Development of independent and functionally equivalent N versions		Diverse Redundancy
developing the comparator and selector unit		Voting Comparison
Implementation Section		
Implementation Description		**Tactic**
The comparator and selector component should be carefully designed to provide an efficient comparison and fast switching.		Voting Comparison
The success of the NSCP Pattern depends on the independent development of the required N versions		Diverse Redundancy
-		-
Tactic Topology Model		

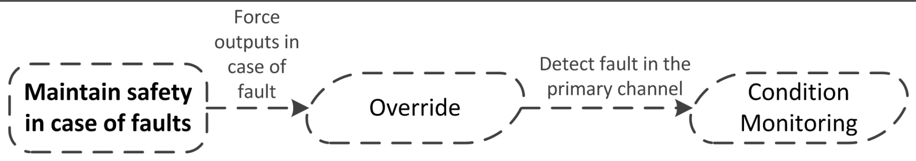


Provide multiple diverse software versions
Diverse Redundancy
Continue operation even in case of faults
Compare two versions within a channel
switch to backup in case of failure
Voting
Comparison

Sanity Check Pattern

Abstract Section		
Core Intent		**Tactic**
The sanity channel, which provides a monitoring to the actuation channel to ensure that the actuation output is approximately correct and within some fixed range.		Override
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
Improve the safety of an embedded system in the presence of single point of failure in a system that includes a fail-safe state and low availability requirement.	A fail-safe state is entered if a primary channel fault is detected. Known hazards in the primary channel can be detected.	Override
Solution Section		
Solution Description		**Tactic**
A safety monitoring method switches the system into its fail-safe state in the presence of failure.		Override
In the case of great difference between the set point and the measured value, the sanity channel forces the actuation channel entering the fail-safe state		Sanity Check
the monitor generates a shutdown signal to the actuation channel		Override
In the case of great difference between the set point and the measured value, the sanity channel forces the actuation channel entering the fail-safe state		Sanity Check Override
Consequences Section		
Consequence Description		**Tactic**
If the result of the comparison shows that the output is totally incorrect and may affect the safety of the system, the monitor generates a shutdown signal to the actuation channel		Override
Implementation Section		
Implementation Description		**Tactic**
The implementation of the monitor component is very simple since it is a very simple unit that includes a simple algorithm to perform the required broad range comparison.		Sanity Check
Tactic Topology Model		
 Force outputs in case of fault Maintain safety in case of faults Override Detect fault in the primary channel Sanity Check		

MONITOR-ACTUATOR PATTERN

Abstract Section		
Core Intent		**Tactic**
A monitoring channel monitors the actuation channel in order to detect and to identify the possible faults		Override
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
Improve the safety of a system that includes a fail-safe state and low availability requirements at reasonable cost.	A fail-safe state is entered if a primary channel failure is detected. Hazards in the primary channel can be detected.	Override
Solution Section		
Solution Description		**Tactic**
The Monitoring Channel monitors the actuation channel to check its proper operation. It takes the information from the set point source and the actuator sensors to detect possible faults in the actuation channel.		Condition Monitoring
In the case of improper operation, it forces the actuation channel to enter the fail-safe state.		Override
The monitor takes the information about the outputs of the actuators, which is collected by the actuator sensors and processed by the monitoring acquisition system, and compares it with the provided set points.		Condition Monitoring
If the result of the comparison shows improper operation in the actuation channel, the monitor generates a shutdown signal to the actuation channel.		Override
Consequences Section		
Consequence Description		**Tactic**
generate the shutdown signal to force the actuation channel entering its fail-safe state		Override
Implementation Section		
Implementation Description		**Tactic**
it is a good idea for the monitoring channel to store some historical information about the monitored value which could be helpful to determine whether the detected value represents a transient or persistent fault.		Condition Monitoring
Tactic Topology Model		
		

Maintain safety in case of faults
Force outputs in case of fault
Override
Detect fault in the primary channel
Condition Monitoring

Watchdog Pattern

Abstract Section	
Core Intent	**Tactic**
The pattern widely used in the embedded systems to make sure that the time-dependent computational processing is proceeding properly as expected in a predefined order	Override

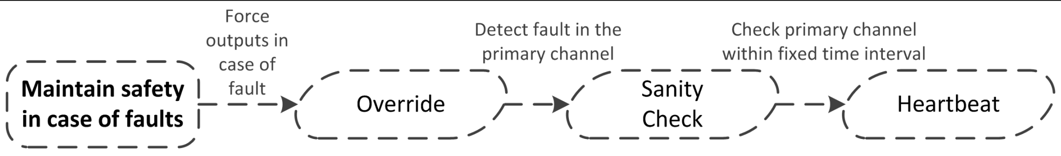
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to make sure that the internal computational processing of the actuation channel is proceeding properly and timely.	A fail-safe state is entered if a primary channel failure is detected. A timing fault in the primary channel can be detected.	Override

Solution Section	
Solution Description	**Tactic**
The watchdog receives liveness messages (Strokes) from the actuation channel on a periodic or in a predefined-sequence base. *Program sequence monitoring*	Heartbeat
The watchdog must be stroked within a specified period of time or it will initiate a corrective action such as a shutdown signal	Override
The Watchdog Pattern checks that the time-dependent computational processing is proceeding properly as expected in a predefined order	Sanity Check
Built In Test (BIT) verifies all or a portion of the internal functionality of the actuation channel.	Sanity Check
Consequently, it issues a shutdown or reset signal to the actuation channel or initiates a corrective action through sending a command signal	Override

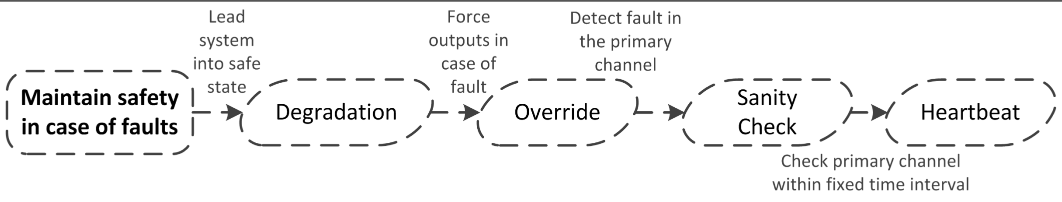
Consequences Section	
Consequence Description	**Tactic**
execution of the built in tests that may be initiated by the watchdog.	Sanity Check

Implementation Section	
Implementation Description	**Tactic**
To increase the fault coverage, it is common to invoke a BIT, CRC, or stack overflow check when the watchdog is stroked to ensure that the computational processing of the actuation channel is proceeding properly.	Sanity Check

Tactic Topology Model

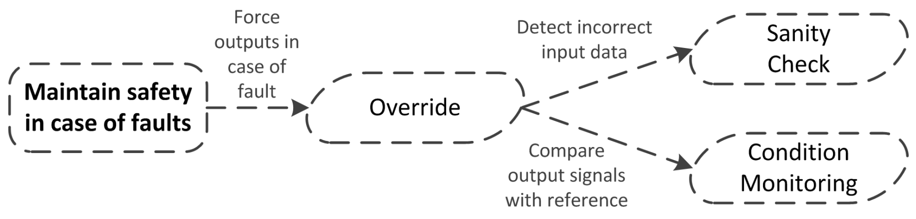

Maintain safety in case of faults
Force outputs in case of fault
Override
Detect fault in the primary channel
Sanity Check
Check primary channel within fixed time interval
Heartbeat

SAFETY EXECUTIVE PATTERN

Abstract Section		
Core Intent		**Tactic**
The safety executive component is responsible for the shutdown of the system as soon as the watchdog sends a shutdown signal		Degradation
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to provide a centralized and consistent method for monitoring and controlling the execution of a complex safety measure in case of failures.	A fail-safe state is entered if a primary channel failure is detected. A timing fault in the primary channel can be detected.	Degradation
Solution Section		
Solution Description		**Tactic**
A centralized safety executive component coordinates all safety-measures required to shut down the system or to switch over to the fail-safe processing channel. *Graceful degradation*		Override Degradation
It is an optional component, which is invoked by the watchdog to run a periodic Built In Test (BIT) to verify all or a portion of the internal functionality of the actuation channel. *Program sequence monitoring*		Sanity Check
The watchdog receives liveness messages (strokes) from the components of the actuation channel in a predefined time frame.		Heartbeat
The Safety Executive tracks and coordinates all safety monitoring to ensure the execution of safety actions.		Degradation
Consequently, it issues a shutdown signal to the safety executive component or initiates a corrective action.		Override
Consequences Section		
Consequence Description		**Tactic**
Execution of the periodic built in tests		Sanity Check
Implementation Section		
Implementation Description		**Tactic**
Graceful degradation		Degradation
The designer should determine whether the new components need to send stroke messages to the watchdog or not		Heartbeat
Tactic Topology Model		
		

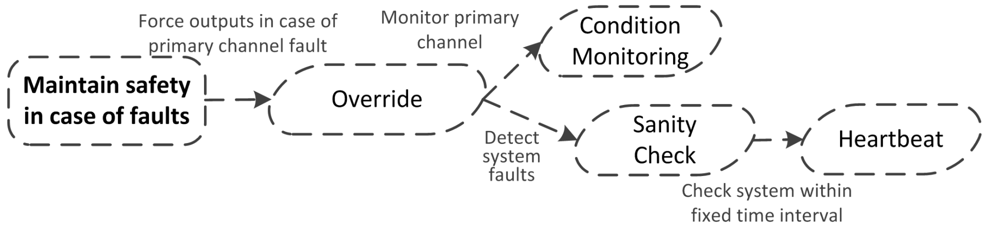
Lead system into safe state
Force outputs in case of fault
Detect fault in the primary channel
Maintain safety in case of faults
Degradation
Override
Sanity Check
Heartbeat
Check primary channel within fixed time interval

Protected Single Channel Pattern

Abstract Section		
Core Intent		**Tactic**
It should be integrated with another safety technique in the presence of immediate fail-safe state to be used for light safety-critical applications.		Override
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to deal with the transient faults to provide some level of safety and reliability to the embedded system in an inexpensive manner	A fail-safe state is entered if a primary channel fault is detected. Hazards in the primary channel can be detected.	Override
Solution Section		
Solution Description		**Tactic**
input data validation and one for actuation monitoring. *Failure detection by online monitoring*		Sanity Check Condition Monitoring
The actuator sensors are used to get feed back signals from the output of the actuators to be used for the actuation monitoring		Condition Monitoring
checks on the input data and the system itself		Sanity Check
Actuator Monitoring: It provides a monitoring to the output of the channel, such as checking the output commands for validity before delivering this command to the actuators. It can also check the output actuators using separate sensors by getting feedback values from the actuators and comparing these values with the previously generated control signals.		Sanity Check Condition Monitoring
Consequently, it can use this information to reconfigure the output processing component to overcome the transient faults when it is possible.		Override
Consequences Section		
Consequence Description		**Tactic**
If the validation checks are performed by hardware, then the extra components are working in parallel with the basic channel which does not affect the basic system in the normal execution		Sanity Check
Implementation Section		
Implementation Description		**Tactic**
In this pattern, there are two versions to be implemented: either (open loop) with only data integrity checking unit, or (close loop) that includes additional actuation monitoring unit.		Sanity Check Condition Monitoring
The existence of a fail-safe state gives the data integrity and the actuator monitoring components the capability to switch the system into the fail-safe state in the presence of persistent fault.		Override
Tactic Topology Model		
		

Maintain safety in case of faults
Force outputs in case of fault
Override
Detect incorrect input data
Sanity Check
Compare output signals with reference
Condition Monitoring

3-Level Safety Monitoring Pattern

Abstract Section		
Core Intent		**Tactic**
The monitoring level monitors the first level, and the control level controls the monitoring level and the entire hardware channel.		Condition Monitoring
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Achieved through Tactic**
How to continue providing the required safety level and to ensure that the system does no injure or harm, when there is any deviation in the output of the actuators from the commanded set point.	A fail-safe state is entered if a primary channel failure is detected. Hazards in the primary channel can be detected A timing fault in the primary/monitor channel can be detected	Condition Monitoring Override
Solution Section		
Solution Description		**Tactic**
The Monitoring Module monitors the actuation module through a comparison of the processing results, input data and the data from the actuatorŠs sensors.		Condition Monitoring
In the case of great difference between the desired value and the measured value, this module forces the actuation channel to switch into its fail-safe state.		Override
If the result of the comparison shows that the output of actuation channel is totally incorrect and may affect the system safety, the monitor will generate a correction command or shutdown signal		Override
Data Validation (Data Integrity Check): It provides a check on the input data during the executing of the desired algorithm to ensure that the input data is valid and in the safe boundaries. *Fault detection and diagnoses*		Sanity Check
A Watchdog is used to provide a sequence control to the monitoring level and to the entire actuation channel. *Program sequence monitoring*		Sanity Check Heartbeat Override
Consequences Section		
Consequence Description		**Tactic**
generate the shutdown or reset signal to force the actuation channel entering the fail- safe state		Override
Implementation Section		
Implementation Description		**Tactic**
If low-sensitive sensors are used for the monitoring level, then a simple algorithm should be used to perform the required broad range comparison.		Condition Monitoring
Tactic Topology Model		
		

Force outputs in case of primary channel fault
Monitor primary channel
Condition Monitoring
Maintain safety in case of faults
Override
Detect system faults
Sanity Check
Heartbeat
Check system within fixed time interval

Appendix References

Babar, M.A. (2007). Improving the Reuse of Pattern-Based Knowledge in Software Architecting. In: *EuroPLoP*. Lero, Ireland, 7–11.

Goble, William M (1998). The Use and Development of Quantitative Reliability and Safety Analysis in New Product Design. PhD thesis. Technical University of Eindhoven.

International Electrotechnical Commission (2010). *IEC 61508, Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety Related Systems*.

Kumar, Kiran and T.V. Prabhakar (2010). Design Decision Topology Model for Pattern Relationship Analysis. In: *1st Asian Conference on Pattern Languages of Programs (AsianPLoP 2010)*.

Varjoranta, Velu (2012). Software safety issues in machine control system design process. PhD thesis. Tampere University of Technology.

Yang, Hao and Xianhui Yang (Aug. 2010). Automatic Generation of Markov Models in Safety Instrumented Systems with Non-identical Channels. In: *2010 International Conference of Information Science and Management Engineering*. IEEE, 287–290.

Zhu, Liming, Muhammad Ali Babar, and Ross Jeffery (2004). Mining Patterns to Support Software Architecture Evaluation. In: *4th Working IEEE / IFIP Conference on Software Architecture (WICSA)*. IEEE.

Security Analysis of Safety Patterns

CHRISTOPHER PRESCHERN, Institute for Technical Informatics, Graz University of Technology
NERMIN KAJTAZOVIC, Institute for Technical Informatics, Graz University of Technology
CHRISTIAN KREINER, Institute for Technical Informatics, Graz University of Technology

Architectural safety patterns provide knowledge about large scale design decisions for safety-critical systems. Safety-critical systems are nowadays increasingly subject to attacks due to their increased connectivity to the Internet. Therefore, we extend existing architectural safety patterns to include security considerations. We apply a STRIDE approach on the safety patterns to obtain relevant threats for each pattern and we structure these threats in a Goal Structuring Notation diagram. We present a catalog of security enhanced safety patterns and we apply one of the patterns to a case study to show how the security-enhanced safety patterns can help for security reasoning.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architecture—*Patterns*; K.4.1 [**Public Policy Issues**] Human Safety; K.6.5 [**Management of computing and information systems**] Security and Protection

ACM Reference Format:

Preschern, C. 2013. Security Analysis of Safety Patterns. Proceedings of PLoP2013. 11 pages.

1. INTRODUCTION

Security concerns are still not sufficiently considered when designing safety-critical systems although they become more relevant due to increasingly interconnected systems. To provide safety engineers with guidelines how to design good systems, safety patterns can be used which describe the safety-related consequences of taking a specific design decision. However, none of the safety patterns in literature extensively cope with the effects of the pattern application on system security.

In this paper we evaluate existing safety patterns regarding their effect on the overall system security. We structurally analyze safety patterns by using the STRIDE approach which is well known in the security domain. This gives us a list of threats for the design patterns which we divide in categories depending on how critical they are for the system's safety. We then present highly critical threats for each pattern in a Goal Structuring Notation diagram, which allows one to easily see which parts of the system are important to protect against attacks. The resulting security enhanced patterns provide a basis for safety engineers to analyze and enhance the security of their systems. We show the application of a pattern and its Goal Structuring Notation diagram in a case study from the substation automation domain.

This paper is structured as follows: Section 2 gives some basic background on the STRIDE approach and on Goal Structuring Notation. Both are used in Section 3 which describes how the security effects of safety patterns are evaluated. Section 4 shows how to apply the security enhanced safety patterns for reasoning about the security of a case study. Section 5 gives related work on security evaluation for design patterns and Section 6 concludes this work. In the Appendix we present our catalog of the security enhanced safety patterns.

2. BACKGROUND

This section gives a basic introduction to the STRIDE threat modeling approach and to Goal Structuring Notation.

2.1 STRIDE Threat Modeling Approach

In order to build a secure system, it is necessary to first find the relevant threats to the system before finding solutions how to mitigate them. The STRIDE approach is a structured way to find these threats. The STRIDE approach was proposed by Microsoft [Howard and LeBlanc 2003] and is nowadays often used as part of security analysis. STRIDE is an acronym, where the letters stand for the six threat categories which are analyzed (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), Elevation of Priviledge (EoP)).

For threat modeling with STRIDE, first a data flow diagram (DFD) has to be constructed. A DFD shows the interaction between system elements and external elements (e.g. users of the system) by graphically presenting all the data flows (inputs/outputs of elements). All relevant STRIDE threats for each element in the diagram are then listed. The relevant threats for different DFD element types are given in Table I.

Table I. STRIDE mapping to DFD element types

DFD element type	S	T	R	I	D	E
External entity	X		X			
Data flow		X		X	X	
Data store		X	X	X	X	
Process	X	X	X	X	X	X

The resulting list of threats can further be elaborated by excluding threats which are not relevant for the specific system and by implementing countermeasures for relevant threats. When all threats are covered, one has a structured argument for system security. We use Goal Structuring Notation to present such a structured argument.

2.2 Goal Structuring Notation

The Goal Structuring Notation (GSN) was developed by [Kelly and Weaver 2004] and is often used in the safety domain for providing a structured argument for the achievement of specific goals. Recently, a standard for the GSN was published which contains definitions of the notation and which presents approaches how to use GSN to elaborate a specific goal [GSN Working Group 2011]. GSN can also be used to argue for system security like in [Cockram and Lautieri 2007]. Figure 1 explains the GSN concepts which are later on used in this paper.

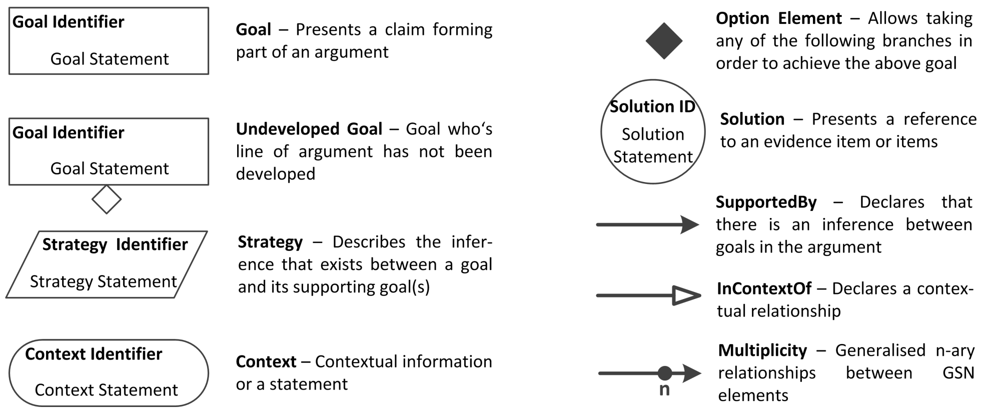

Goal Identifier
Goal Statement
Goal – Presents a claim forming part of an argument
Goal Identifier
Goal Statement
Undeveloped Goal – Goal who's line of argument has not been developed
Strategy Identifier
Strategy Statement
Strategy – Describes the inference that exists between a goal and its supporting goal(s)
Context Identifier
Context Statement
Context – Contextual information or a statement
Option Element – Allows taking any of the following branches in order to achieve the above goal
Solution ID
Solution Statement
Solution – Presents a reference to an evidence item or items
SupportedBy – Declares that there is an inference between goals in the argument
InContextOf – Declares a contextual relationship
n
Multiplicity – Generalised n-ary relationships between GSN elements

Fig. 1. GSN concepts used in this paper taken from [GSN Working Group 2011]

To show how a GSN goal is achieved, it is linked to an argument (GSN strategies, GSN subgoals) which ends up in the evidence (GSN solution) supporting the claim that the goal is achieved. Figure 2 shows an example for the application of GSN. The main goal in the example is that an attacker cannot obtain some confidential data. In the next step, context elements are added which say that the data is locally stored on a computer and transmitted to another computer. The main goal is split up into the subgoals to protect the stored data and the transmitted data. Protecting transmitted data is achieved by just transmitting the data over a protected TLS channel (GSN strategy). For this TLS channel, we need evidence that it works properly. This evidence (GSN solution element) is that the used implementation is security certified. Protecting stored data is an undeveloped goal which means that the security argument for this subgoal is not yet complete and further arguments have to be included here in order to obtain a complete argument that the overall goal (protecting the confidential data) is achieved. In the example, GSN provides a structured way to show how the rather unspecific goal to protect confidential data is (partially) achieved by specific measures (the TLS channel).

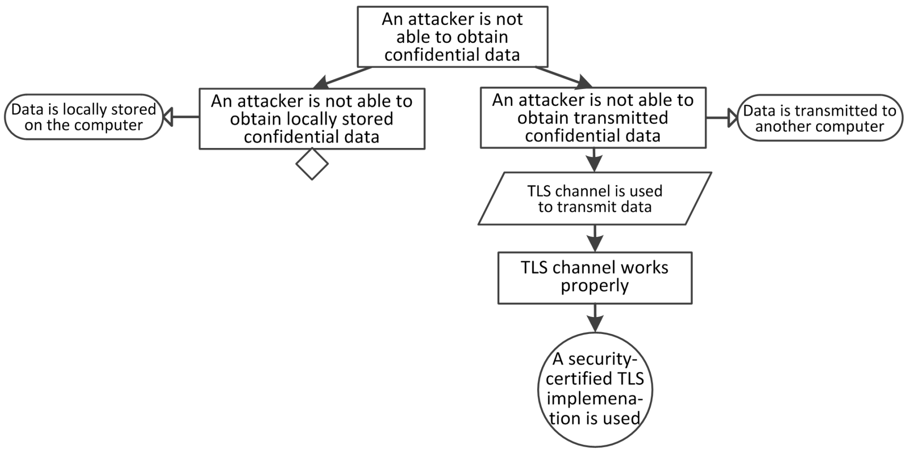

An attacker is not able to obtain confidential data
Data is locally stored on the computer
An attacker is not able to obtain locally stored confidential data
An attacker is not able to obtain transmitted confidential data
Data is transmitted to another computer
TLS channel is used to transmit data
TLS channel works properly
A security-certified TLS implemena-tion is used

Fig. 2. GSN example showing a security argument

2.3 Alternative Methods for Threat Elaboration

STRIDE is a very generic security analysis and can be used as a starting point to develop security requirements and security countermeasures. In this work we build a GSN diagram to argue for STRIDE threat mitigation; however, instead of GSN several alternatives could be used:

- The Secure Tropos project [Mouratidis and Giorgini 2007] provides a tool to model a system and to analyze its threats with the STRIDE method [Rojas and Mahdy 2011]. Threat mitigation mechanisms can be added to the system model and security reports can be printed with the tool. The reason for using GSN diagrams instead to the Secure Tropos model representation including tooling support is that the GSN notation is well known in the safety domain.
- Microsoft's security development lifecycle suggests to build threat trees for each STRIDE threat and to mitigate each element of such a threat tree. The reason for using GSN diagrams instead of threat trees is that GSN diagrams easily allow to integrate security countermeasures into the notation and to further analyze STRIDE threats for these countermeasures. With the threat tree notation that would be cumbersome.
- Fault Trees can also be used to integrate security threats [Nai-Fovino et al. 2009]. However, The reason for choosing GSN above fault trees is that GSN is already known and applied in the safety and security domain whereas fault trees are usually just used in the safety domain.

3. ENHANCING PATTERNS WITH SECURITY REASONING

In this section we present and apply the approach how to use the STRIDE analysis for safety patterns in order to obtain a GSN argument for the patterns which helps to identify threats and to argue for the security of a system which applies a pattern.

3.1 Getting the Data Flow Diagram

The catalog of safety architecture patterns presented in [Preschern et al. 2013] shows several safety patterns with a consistent notation. Each of the patterns describes how a *Basic System* consisting of hardware or software elements can be modified (e.g. through adding a watchdog, or through replication) in order to increase its safety. Each of the patterns provides a diagram which shows the hardware and software elements of the pattern and their interaction. For the patterns, this diagram contains all the necessary information for the STRIDE analysis and will be used instead of a data flow diagram.

Figure 3 shows such a diagram for the *Basic System* (to which the patterns from [Preschern et al. 2013] can be applied) The *Basic System* gets input data, processes that input data in the primary channel, and produces output data for a safety-critical process.

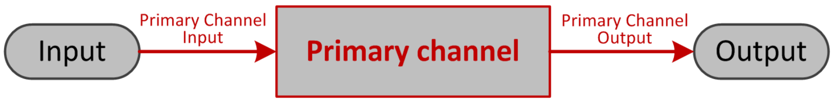


Fig. 3. Basic system which is the starting point for the safety patterns

3.2 Getting the Threats

By using an adapted STRIDE approach, we analyze the pattern diagrams to list the security threats for each of the patterns.

We just consider two element types for the STRIDE analysis: *Data flows* and *Processing elements*. For both types, we omit the threats Repudiation and Information Disclosure, because they do not directly influence the safety functionality of a system. Furthermore, for the *Processing elements*, we omit the Tampering and Denial of Service threats, because an attacker usually has no access to processing elements which perform safety-critical functionality. Therefore, he needs to elevate his privileges before starting a tampering or DoS attack on a processing element. Our resulting relevant threats for the pattern diagram element types are shown in Table II.

Table II. STRIDE mapping to safety pattern element types

DFD element type	**S**	**T**	**R**	**I**	**D**	**E**
Data flow		X			X	
Processing element	X					X

With this mapping of relevant threats, we go through each element of the pattern diagram to obtain a list of relevant threats for the pattern. For the *Basic System* we get the following list of threats:

- Tampering of Primary Channel input
- DoS against Primary Channel input
- Spoofing of the Primary Channel
- EoP on the Primary Channel
- Tampering of Primary Channel output
- DoS against the Primary Channel output

3.3 Categorizing the Threats

For each pattern we divide the obtained threats into criticality categories which make it easier to quickly see which threats are especially relevant for the pattern. The threats are categorized as:

- Threats to the safety-critical functionality of the system
- Threats which can bring the system into a safe state (e.g. shut it off)
- Threats which do not directly influence the system functionality and leave the system fully functional

To determine which category a threat belongs to, we analyze what would happen if a successful attack related to the threat was applied. If the attack could arbitrarily modify the system's output data, then the threat is **safety-critical**. If the attack could shut the system off, then the threat is classified as one which **leads to a safe state**. If the attack does not influence the system's output, the threat is classified as one where the **system remains fully functional**.

We display the categorized threats in a table which lists them according to their STRIDE type and criticality category. All the threats for the *Basic System* are underlines and printed in green color. All other threats (threats for a safety pattern apart from the *Basic System* threats), are printed in black color. This has the advantage that for the safety patterns in the Appendix, one can easily see to which criticality category the *Basic System's* threats are shifted when applying the pattern or whether the *Basic System's* threats are then even relevant anymore.

For the *Basic System*, after applying the described threat categorization, we obtain the threat table shown in Table III. We can see that all threats are categorized as safety-critical. For example, the *"Tampering of Primary Channel input"* threat is safety-critical, because if someone can maliciously modify the Primary Channel input data, then, in general, it is also possible to modify the system's output data, because the output data calculation of the primary channel depends on the input data. We can also see that all threats underlined and printed in green. This is, because per definition, we print all *Basic System* threats underlined and in green. When looking at the patterns in the Appendix, also threats printed in black are present and the advantage of using different colors in the table can be seen, because one can easily see which basic threats (underlined, green) are shifted into other columns. This gives a quick overview of how the pattern affects the existing threats.

Table III. STRIDE threats relevant for the *Basic System*

	safety-critical	**leads to a safe state**	**system remains fully functional**
S	Spoofing of the Primary Channel	-	-
T	Tampering of Primary Channel input Tampering of Primary Channel output	-	-
R	-	-	-
I	-	-	-
D	DoS against Primary Channel input DoS against Primary Channel output	-	-
E	EoP on Primary Channel	-	-

To highlight the safety-critical threats, we color all components in the pattern's diagram which are related to safety-critical threats in red. For the *Basic System*, this was already done in Figure 3. For the patterns presented in the Appendix, this makes it very easy to get a first impression of which components especially have to be protected. All the patterns in the Appendix contain a table with their categorized threats.

3.4 Constructing the Security GSN

In some cases, threats which are not classified as safety-critical can become part of an attack affecting system safety if they are combined. To also capture thi s information we construct a GSN diagram for each pattern. The top-level GSN goal is *to maintain the safety functionality even in case of an attack*. The subgoals are the prevention of attacks leading to the analyzed safety-critical threats or the prevention of attack combinations[1].

Using GSN diagrams to represent attacks is not the original approach presented by [Howard and LeBlanc 2003] for the STRIDE method. The original approach is to gather attacks and use a tree-like notation (called attack trees) to display how these attacks can be combined to form STRIDE threats. However, attack trees just capture the information how to relate attacks and do not contain information about the countermeasures against these attacks. With GSN it is possible to relate countermeasures (GSN strategies) to the attack which they mitigate (GSN goals). Thus, compared to attack trees, GSN diagrams bring the advantage of establishing a link between the security goals (protect against system threats) and the implemented countermeasures. A similar approach was already suggested by [Moleyar and Miller 2007].

Figure 4 shows the security GSN diagram for the *Basic System* which is rather straightforward, because all its threats are safety-critical. However, if we would construct a GSN for a system similar to the *Basic System* but which additionally has a safe state when it is shut off, we would obtain a slightly different GSN diagram. If the system had a safe state when shut off, the DoS threats would not be safety critical, but they would belong to the second column ("leads to a safe state") in Table III. For the GSN diagram this would mean that the DoS threats would not be part of it, because they cannot lead the system to a critical state (also not if both DoS threats would be combined).

All of the patterns in the Appendix contain a security GSN diagram. These diagrams are more complex than the diagram in Figure 4 and yield additional information regarding the possible threat combinations which are safety-critical. Such a GSN diagram can then be used as a basis for security reasoning for a specific architecture which applies one of the safety patterns. The GSNs of the patterns contain undeveloped goals, because the implementation details for a specific architecture applying one of the patterns are not yet known. These undeveloped goals have to be developed (by adding architecture-specific claims and proves that support the goal) to obtain a complete security argumentation.

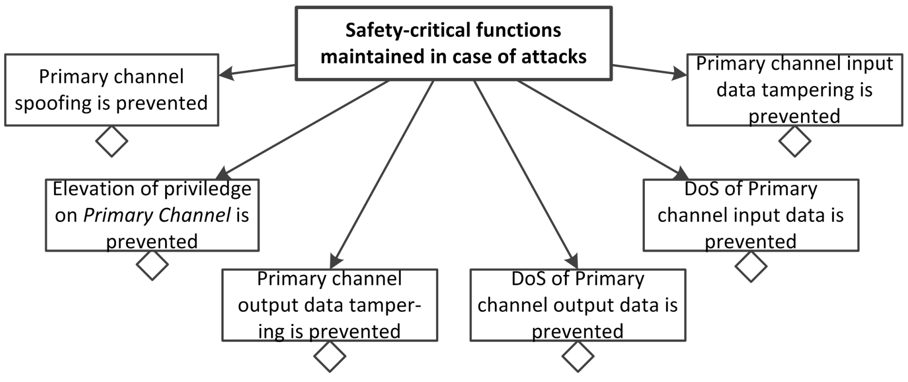

Safety-critical functions maintained in case of attacks
Primary channel spoofing is prevented
Primary channel input data tampering is prevented
Elevation of priviledge on Primary Channel is prevented
DoS of Primary channel input data is prevented
Primary channel output data tampering is prevented
DoS of Primary channel output data is prevented

Fig. 4. Security GSN diagram for the basic system

[1]To model a combination of attacks, the subgoals would be related with a GSN option element to the main goal - the HETEROGENOUS DUPLEX PATTERN is an example which contains a combined attack and therefore makes use of the GSN option element

4. APPLYING THE SECURITY ENHANCED SAFETY PATTERNS TO A CASE STUDY

In this section we apply one of the safety patterns from the Appendix to a case study. With the security analysis of the safety pattern, we construct a complete security argument for the system architecture.

4.1 System description

In our case study we apply the HETEROGENOUS DUPLEX PATTERN to an electrical substation automation device. Substations handle functions like voltage protection and conversion between different voltage distribution networks. The substation automation device in our case study is a safety-critical component which handles over-voltage protection. Based on measured current and voltage input values, the device has to decide if a power distribution network should be cut off in order to protect other devices from over-voltage. The over-voltage protection device obtains its sensor inputs from an IEC61850 merging unit, which is a sensor unit distributing sensor values via Ethernet. Based on this sensor data, the system has to control actuators which are hardwired to the device. The system is connected to the local substation Ethernet network to enable firmware updates.

Figure 5 gives an overview of the system architecture after applying the HETEROGENOUS DUPLEX PATTERN (more details about the pattern are given in the Appendix on page 11). The substation automation device has two CPUs where each CPU input is supplied with its own set of sensor data. To compute the actuator output value, the CPUs run diverse software versions. This means the software versions have the same functionality, but different implementations. Each CPU runs a diagnostic test and periodically sends the results of the test to an FPGA which checks the diagnostic results and switches the actuator output to the backup CPU output if the diagnostic test of the primary CPU fails. An external connection to both CPUs can be established via the local Ethernet to install firmware updates on the CPUs.

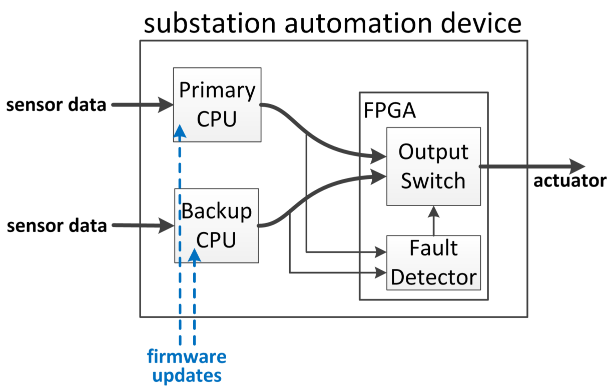


Fig. 5. Substation automation device architecture

The architecture is very similar to the basic HETEROGENOUS DUPLEX PATTERN which is described in the Appendix. The only differences are that the architecture has an additional connection to the CPUs for firmware updates and that the fault detector and the output switch are realized on a single hardware component.

4.2 Adapting the security GSN from the pattern

The HETEROGENOUS DUPLEX PATTERN includes a security GSN diagram which captures the aim to mitigate safety-relevant threats for this pattern as subgoals. These subgoals are undeveloped GSN goals (because the GSN diagram of the pattern does not yet include information how these subgoals are achieved). We now want to develop the subgoals in order to obtain a complete security argument for our architecture. We go through every undeveloped goal and check whether the threat is actually a threat for the specific architecture. If it is not, we add the information why it is not a relevant threat to the GSN notation. If it is a relevant threat, we suggest mitigation strategies. Figure 6 shows the resulting security GSN diagram for the substation automation device architecture. Black elements with solid lines are taken from the GSN diagram of the HETEROGENOUS DUPLEX PATTERN and green, dashed elements are added for the specific architecture.

The completed GSN diagram in Figure 6 shows us that some of the threats to the system (e.g. the *"DoS of Fault Detector is prevented"* GSN goal element) are irrelevant. However, we do not eliminate these elements from the diagram, but add GSN elements which argue why these threats are sufficiently handled by the architecture itself (e.g. *"An attacker has no physical access to the Fault Detector"* GSN context element and *"The Fault Detector is hardwired to the CPU diagnosis output"* GSN solution element).

Some other threats are not irrelevant but require countermeasures. For example, to mitigate the EoP threats to the switch, to the fault detector, and to the CPUs, the countermeasure to thoroughly test these units (GSN strategy elements) and to provide the test results (GSN solution elements) is applied. Additionally, for the CPUs, the countermeasure to check the integrity of firmware updates is applied to handle the threat of achieving EoP on the CPU by using a malicious firmware update. Another set of threats which have to be mitigated with appropriate countermeasures, are threats to the merging unit. These threats are countered by putting the merging unit into a separate Ethernet network to which an attacker does not have access.

4.3 Benefits of the Security GSN diagram

The main benefit of the GSN diagram is that with the application of a safety architecture pattern, we get a structured representation of relevant security threats. This allows us on the one hand to argue for the overall system security and on the other hand points to weaknesses of the architecture. By not deleting irrelevant threats but adding information to the GSN diagram why these threats are irrelevant, we obtain a security argument for the architecture which is complete regarding its safety-relevant STRIDE threats.

5. RELATED WORK

This section covers related work on the security evaluation of safety-critical systems and on the security evaluation of design patterns.

[Hansen 2009] presents a security analysis of a safety-critical automation device which highlights attacks compromising the system safety. [Johnson and Yepez 2011a] and [Johnson and Yepez 2011b] presents a combined security and safety risk assessment methodology where security and safety arguments are shown in a GSN diagram. Security threats are analyzed for a case study and the threats are included in an existing safety GSN to obtain a unified assurance case for safety and security. [Nai-Fovino et al. 2009] present a method to integrate security reasoning into fault trees. They discuss how to analyze the risk of security aspects in order to integrate their probabilities consistently into the fault tree notation. A similar apprach is taken by [Ugljesa and Wacker 2011] to integrate security considerations into the error probability calculation of a 2oo4 architecture[2]. [Yampolskiy et al. 2012] present an extension of data flow diagrams which allows analyzing an architecture for STRIDE attacks as well as for safety.

[Yautsiukhin and Scandariato 2008] conduct a STRIDE analysis for a case study and discuss how well several patterns can counter the threats. They use a risk assessment method to rate the threat severity and they assign a value to each pattern describing how well the pattern copes with different threats. With this method the security of different patterns for a system can be quantitatively compared. A similar approach is taken in [Halkidis et al. 2006b], [Halkidis et al. 2006a], and [Halkidis et al. 2008]. They evaluate the effectiveness of web security patterns against STRIDE attacks by experiments. With these results they suggest patterns for a web system by first conducting a STRIDE analysis for the concrete system and then suggesting the patterns which mitigate the STRIDE attacks best. This work is also done for security patterns in general in [Halkidis et al. 2004], where a mapping between

[2]The 2oo4 architecture is a special version of the M-OUT-OF-N PATTERN which is explained on page 11.

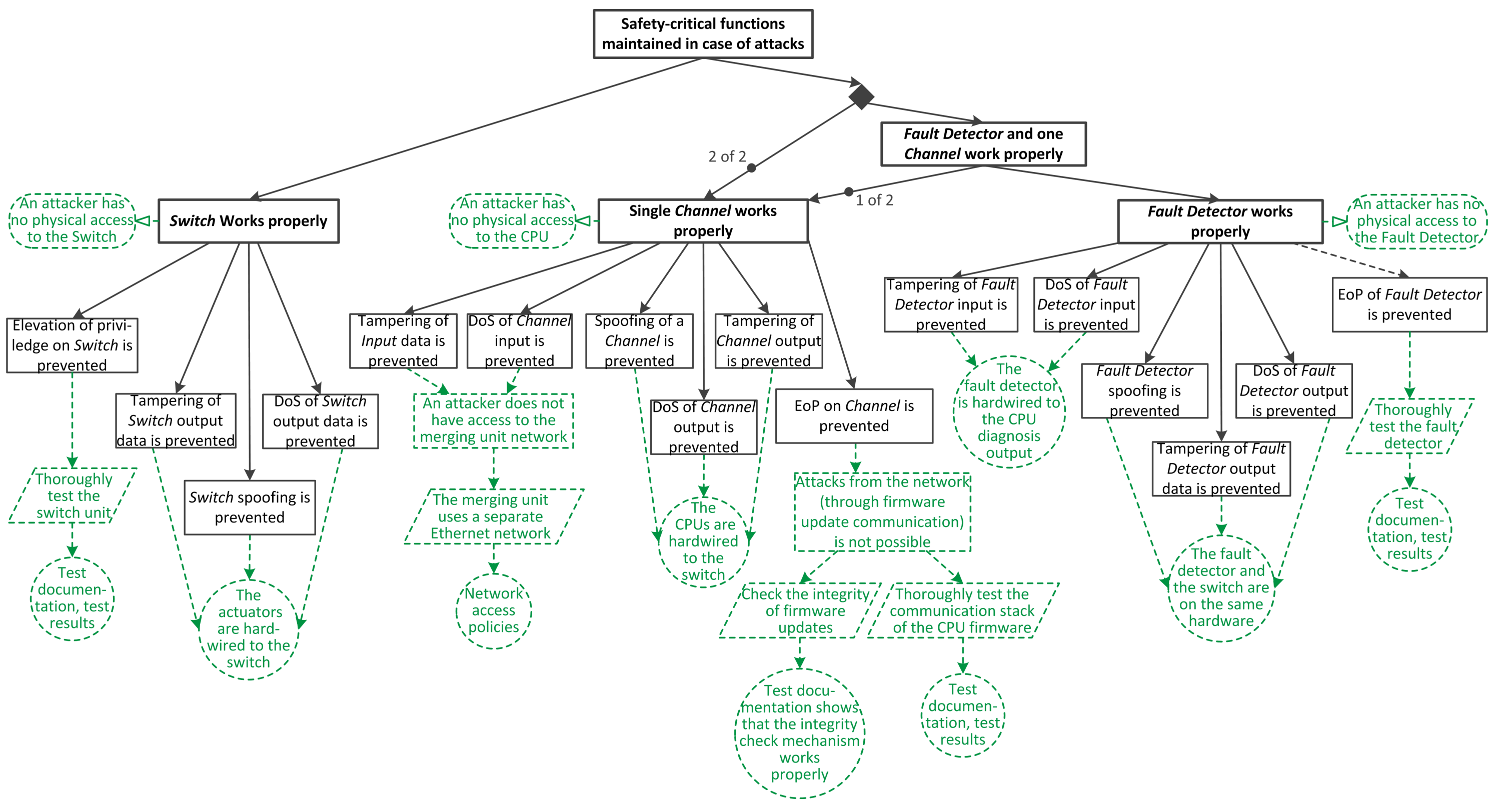

Safety-critical functions maintained in case of attacks
Fault Detector and one Channel work properly
2 of 2
1 of 2
Switch Works properly
Single Channel works properly
Fault Detector works properly
An attacker has no physical access to the Switch
An attacker has no physical access to the CPU
An attacker has no physical access to the Fault Detector
Elevation of privi-ledge on Switch is prevented
Tampering of Switch output data is prevented
Switch spoofing is prevented
DoS of Switch output data is prevented
Tampering of Input data is prevented
DoS of Channel input is prevented
Spoofing of a Channel is prevented
Tampering of Channel output is prevented
DoS of Channel output is prevented
EoP on Channel is prevented
Tampering of Fault Detector input is prevented
DoS of Fault Detector input is prevented
Fault Detector spoofing is prevented
Tampering of Fault Detector output data is prevented
DoS of Fault Detector output is prevented
EoP of Fault Detector is prevented
Thoroughly test the switch unit
Test documen-tation, test results
The actuators are hard-wired to the switch
An attacker does not have access to the merging unit network
The merging unit uses a separate Ethernet network
Network access policies
The CPUs are hardwired to the switch
Attacks from the network (through firmware update communication) is not possible
Check the integrity of firmware updates
Thoroughly test the communication stack of the CPU firmware
Test docu-mentation shows that the integrity check mechanism works properly
Test documen-tation, test results
The fault detector is hardwired to the CPU diagnosis output
The fault detector and the switch are on the same hardware
Thoroughly test the fault detector
Test documen-tation, test results

Fig. 6. Substation automation device security argument (based on the GSN of the HETEROGENOUS DUPLEX PATTERN)

several security patterns and their effectiveness for STRIDE attacks is presented. In [Schaad and Borozdin 2012] and [Schaad and Garaga 2012] a tool is presented which reports threats for an architecture by automatically applying the STRIDE analysis to an architecture model. As in our approach, the STRIDE analysis is adapted to just include the threats relevant for the specific architecture element types. [Hamid et al. 2010] take another approach with the TERESA project, by applying a model-based approach to integrate design patterns in order to argue about the safety and security of a system. The tool-based process of how to apply the design patterns is described in [Hamid et al. 2013].

6. CONCLUSION

In this paper we added a GSN diagram describing security threats to safety architecture patterns and we discussed the application of these security enhanced safety patterns to a case study.

The safety patterns described in the Appendix all provide a data flow diagram. Therefore, it is easy to analyze the security of the safety patterns by using the STRIDE approach. All of the described safety patterns enhance the same basic system which makes it possible to compare the security attributes of the different patterns.

The main benefits of the security GSN diagram are that it provides a structured argument for the security of a safety system and that it indicates security flaws of the design. Another important benefit of the security enhanced patterns is that safety experts who use these patterns are confronted with the STRIDE approach. This increases the awareness of security threats in the safety domain which is in our opinion not sufficiently addressed so far.

ACKNOWLEDGMENTS

We would like to thank our shepherd Robert Hanmer who significantly helped to improve this paper by providing us with very helpful feedback on the paper structure and contents as well as on its style.

REFERENCES

Cockram, T. J. and Lautieri, S. R. 2007. Combining Security and Safety Principle in Practice. In *2nd Institution of Engineering and Technology International Conference on System Safety*. IEEE, 159–164.

GSN Working Group. 2011. GSN Community Standard Version 1. http://www.goalstructuringnotation.info/.

Halkidis, S., Chatzigeorgiou, A., and Stephanides, G. 2004. A qualitative evaluation of security patterns. In *6th International Conference on Information and Communications Security*. Springer, 132–144.

Halkidis, S., Tsantalis, N., Chatzigeorgiou, A., and Stephanides, G. 2008. Architectural Risk Analysis of Software Systems Based on Security Patterns. *IEEE Transactions on Dependable and Secure Computing 5,* 3, 129–142.

Halkidis, S. T., Chatzigeorgiou, A., and Stephanides, G. 2006a. A qualitative analysis of software security patterns. *Computers & Security 25,* 5, 379–392.

Halkidis, S. T., Chatzigeorgiou, A., and Stephanides, G. 2006b. Quantitative Evaluation of Systems with Security Patterns Using a Fuzzy Approach. In *Proceedings of the 2006 international conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET*. Springer, 554–564.

Hamid, B., Desnos, N., Grepet, C., and Jouvray, C. 2010. Model-based security and dependability patterns in RCES - the TERESA Approach. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems - S&D4RCES '10*. ACM Press, New York, New York, USA.

Hamid, B., Geisel, J., Ziani, A., Bruel, J.-m., and Perez, J. 2013. Model-Driven Engineering for Trusted Embedded Systems Based on Security and Dependability Patterns. In *16th International SDL Forum*. Springer, 72–90.

Hansen, K. 2009. Security attack analysis of safety systems. *IEEE Conference on Emerging Technologies & Factory Automation*, 1–4.

Howard, M. and LeBlanc, D. 2003. *Writing Secure Code*. Microsoft Press.

Johnson, C. W. and Yepez, A. A. 2011a. Cyber Security Threats to Safety-Critical Space-Based Infrastructures. In *Proceedings of the Fifth Conference of the International Association for the Advancement of Space Safety*. Number 1.

Johnson, C. W. and Yepez, A. A. 2011b. Mapping the Impact of Security Threats on Safety-Critical Global Navigation Satellite Systems. In *Proceedings of the 29th International Systems Safety Society*. Number 1. International Systems Safety Society.

Kelly, T. and Weaver, R. 2004. The Goal Structuring Notation Ű A Safety Argument Notation. In *Proceedings of the Dependable Systems and Networks Conference*.

MOLEYAR, K. AND MILLER, A. 2007. Formalizing attack trees for a SCADA system. In *International Conference on Critical Infrastructure Protection*. IFIP.

MOURATIDIS, H. AND GIORGINI, P. 2007. Secure Tropos: A Security-Oriented Extension of the Tropos Methodology. *International Journal of Software Engineering and Knowledge Engineering 17,* 2, 23–36.

NAI-FOVINO, I., MASERA, M., AND DE-CIAN, A. 2009. Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety 94,* 9, 1394–1402.

PRESCHERN, C., KAJTAZOVIC, N., AND KREINER, C. 2013. System of safety-critical embedded Architecture Patterns. In *EuroPLoP*.

ROJAS, D. M. AND MAHDY, A. M. 2011. Integrating Threat Modeling in Secure Agent-Oriented Software Development. *International Journal of Software Engineering 2,* 2, 23–36.

SCHAAD, A. AND BOROZDIN, M. 2012. TAM2: Automated Threat Analysis. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 1103–1108.

SCHAAD, A. AND GARAGA, A. 2012. Automating architectural security analysis. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*. ACM, 131–132.

UGLJESA, E. AND WACKER, H.-D. 2011. Modeling Security Aspects in Safety Environment. In *7th International Conference on Electrical and Electronics Engineering*. 46–50.

YAMPOLSKIY, M., HORVATH, P., KOUTSOUKOS, X. D., XUE, Y., AND SZTIPANOVITS, J. 2012. Systematic analysis of cyber-attacks on CPS-evaluating applicability of DFD-based approach. In *5th International Symposium on Resilient Control Systems*. IEEE, 55–62.

YAUTSIUKHIN, A. AND SCANDARIATO, R. 2008. Towards a quantitative assessment of security in software architectures. In *13th Nordic Workshop on Secure IT Systems (NordSec)*.

A. SECURITY ENHANCED SAFETY PATTERNS

This publication has been shortened and this section has been skipped, because it contains the same information as present in Appendix A in Publication 4: "Safety Architecture Pattern System with Security Aspects". The full publication of the current paper including this section is available at the ACM Digital Library.

Safety Architecture Pattern System with Security Aspects

Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner

Institute for Technical Informatics, Graz University of Technology, Graz, Austria
christopher.preschern@tugraz.at, nermin.kajtazovic@tugraz.at, christian.kreiner@tugraz.at

Abstract. This article builds a structured pattern system with safety patterns from literature and presents the safety patterns. The patterns are analyzed regarding their basic safety-related design decisions (safety tactics) and relationships between the patterns are structurally developed based on these safety tactics. To analyze security aspects, the STRIDE security analysis is used to list relevant threats for the patterns. The threats and the safety tactics are represented in Goal Structuring Notation diagrams as part of the patterns to enable security and safety reasoning.

Keywords: architecture patterns, safety, security, goal structuring notation, STRIDE analysis

1 Introduction

Increasing connectivity of embedded systems makes the influence of security aspects even for safety-critical systems more and more relevant. However, rather often safety experts are not familiar with the field of security.

To provide safety architects with a starting point how to bring security into their system and to provide them with good solutions for safety architectures, we build a safety architecture pattern system[1] including a security analysis for the patterns. We analyze safety patterns for basic design decisions (safety tactics) which are applied in the patterns. Based on the applied safety tactics we find relationships between the patterns. For example, all patterns applying the *Voting* tactic, are likely to be related. Additionally we use the tactics to build a Goal Structuring Notation (GSN) diagram which presents how a pattern achieves its safety goal by applying these tactics. Furthermore, we relate the safety tactics to the IEC 61508 safety certification standard to build a GSN diagram which allows to reason how the overall architecture is related to safety methods described in the standard.

[1] A "pattern system" is similar to a "pattern language", but compared to a pattern language it does not claim to be complete (Buschmann et al., 1996). Precise definitions about the difference between pattern collections/systems/languages can be found in (Schumacher, 2003)

From a security point of view, we analyze all safety patterns with the STRIDE security method. STRIDE results in a list of relevant threats for the pattern which we structurally present in a GSN diagram.

Section 2 of this article provides related work on safety patterns, on how to organize safety patterns, and on security analysis of safety systems and patterns in particular. Section 3 provides basics on safety tactics, GSN, and the STRIDE analysis. These basics are required to understand Section 4 which integrates an example pattern into our pattern system. Section 5 applies a safety pattern to a case study and Section 6 concludes this work. Appendix A presents all the patterns of the safety pattern system and Appendix B presents safety tactics.

2 Related Work

2.1 Safety Patterns

Table 1 gives an overview of literature presenting safety patterns.

Title	**Description**
(Daniels et al., 1997) *"The Reliable Hybrid Pattern - A Generalized Software Fault Tolerant Design Pattern"*	A pattern which includes software fault tolerance techniques (e.g. N-version programming, voting, acceptance test) is presented. The pattern is presented as a generic architecture which explicitly states decision alternatives in the pattern.
(Douglass, 1998) *"Safety-Critical System Design"*	The article covers safety architecture patterns and discusses how they can be implemented.
(Saridakis, 2002) *"A System of Patterns for Fault Tolerance"*	This paper introduces several architectural fault-tolerance patterns and discusses how to group them.
(Douglass, 2002) *"Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems"*	Besides other patterns, this book covers safety-related architecture patterns and also includes the patterns from (Douglass, 1998).
(Grunske, 2003) *"Transformational Patterns for the Improvement of Safety Properties in Architectural Specification"*	This paper presents patterns for architecture transformations to increase the overall system safety. Some of the patterns are related to the patterns from (Douglass, 2002).
(Hanmer, 2007) *"Patterns for Fault Tolerant Software"*	The book provides a pattern language of fault-tolerance patterns grouped as error detection, error processing, error mitigation, fault treatment, and architectural patterns.
(Douglass, 2010) *"Design Patterns for Embedded Systems in C"*	The book presents design patterns implemented in C. Some safety-related patterns come from (Douglass, 2002).
(Armoush, 2010) *"Design Patterns for Safety-critical Embedded Systems"*	This PhD thesis introduces new and collects existing safety patterns for embedded systems (mostly based on (Douglass, 2002) for hardware and (Pullum, 2001) for software patterns).
(Hampton, 2012) *"Survey of Safety Architectural Patterns"*	This survey presents the application of the patterns from (Armoush, 2010) within a company. Furthermore, some new and rather domain-specific safety patterns are introduced.
(Rauhamäki et al., 2012) *"Architectural Patterns for Functional Safety"*	The paper presents 4 patterns related to separating the safety functionality from non-critical functionality.
(Rauhamäki et al., 2013) *"Patterns for Safety and Control System Cooperation"*	The paper presents 3 safety patterns for control systems.
(Rauhamäki and Kuikka, 2013) *"Patterns for Controlling System Safety"*	The paper presents 4 safety patterns for control systems.
(Powel, 2013) *"Software Design Architecture Patterns for Embedded Systems"*	The book chapter discusses some patterns introduced in (Douglass, 1998).

Table 1: Literature which introduces safety-related patterns

2.2 Organizing Safety Patterns

(Saridakis, 2002) presents several fault-tolerance patterns in detail and discusses how they can be related to each other. The patterns are classified according to several criteria: pattern complexity, space requirements, time requirements, failure types which are handled by the pattern, and the pattern aim (error detection, recovery, or masking). (Hanmer, 2007) also describes fault-tolerance patterns and presents the patterns and their relationships as a pattern language.

(Armoush, 2010) provides in his PhD thesis a comprehensive collection of safety architecture patterns for embedded systems. Most of the patterns are taken from literature and all are presented in a common pattern format. However, the relationships between the patterns are not described in detail. Armoush provides a tool which lists the patterns and provides detailed information about them (e.g. reliability calculations) when selected.

To bridge the gap between the high-level safety pattern descriptions and their actual implementation, (Gawand et al., 2011) represent safety patterns in UML notation. This is also done by (Sarma et al., 2013) with the pattern catalog of (Armoush, 2010). This idea was taken further by (Antonino et al., 2012) who introduce a safety-related UML profile to capture architectural safety pattern elements (e.g. voter) and to define rules for them. Based on this idea (Olivera, 2012) implements a repository for safety patterns including their UML notation.

2.3 Security Analysis of Design Patterns and Safety Systems

(Yautsiukhin and Scandariato, 2008) conduct a STRIDE analysis for a case study and discuss how well several patterns can counter the threats. They use a risk assessment method to rate the threat severity and they assign a value to each pattern describing how well the pattern copes with different threats. With this method the security of different patterns for a system can be quantitatively compared. A similar approach is taken in (Halkidis et al., 2006a), (Halkidis et al., 2006b), and (Halkidis, Tsantalis, et al., 2008). They evaluate the effectiveness of web security patterns against STRIDE attacks by experiments. With these results they suggest patterns for a web system by first conducting a STRIDE analysis for the concrete system and then suggesting the patterns which mitigate the STRIDE attacks best. This work is also done for security patterns in general in (Halkidis et al., 2004), where a mapping between several security patterns and their effectiveness for STRIDE attacks is presented. In (Schaad and Borozdin, 2012) and (Schaad and Garaga, 2012), a tool is presented which reports threats for an architecture by automatically applying the STRIDE analysis to an architecture model. As in our approach, the STRIDE analysis is adapted to just include the threats relevant for the specific architecture element types. (Hamid, Desnos, et al., 2010) take another approach with the TERESA project, by applying a model-based approach to integrate design patterns in order to argue about the safety and security of a system. The tool-based process of how to apply the design patterns is described in (Hamid, Geisel, et al., 2013).

Unrelated to design patterns, but related to security evaluation of safety systems in general, (Hansen, 2009) analyzes a safety-critical automation device and highlights attacks compromising system safety. (Johnson and Yepez, 2011a) and (Johnson and Yepez, 2011b) present a combined security and safety risk assessment methodology where security and safety arguments are shown in a GSN diagram. Security threats are analyzed for a case study and the threats are included in an existing safety GSN to obtain a unified assurance case for safety and security. (Nai-Fovino et al., 2009) present a method to integrate security reasoning into fault trees. They discuss how to analyze the risk of security aspects in order to integrate their probabilities consistently into the fault tree notation. This idea is further elaborated in (Steiner and Liggesmeyer, 2013) where a safety and security analysis is performed based on component fault trees. A similar approach is taken by (Ugljesa and Wacker, 2011) to integrate security considerations into the error probability calculation of a 2oo4 architecture[2]. A detailed security analysis with security enhancements for the 1oo2 architecture is shown in (Preschern et al., 2012a). (Yampolskiy et al., 2012) present an extension of data flow diagrams which allows analyzing an architecture for STRIDE attacks as well as for safety.

3 Basics

3.1 Architectural Tactics

Tactics are architectural design decisions which influence and manipulate quality attributes (Bachmann et al., 2003). Compared to design patterns, they describe general concepts or principles and do not describe solutions for a problem in a given context. For example, the *Voting* safety tactic describes the idea how to achieve failure containment by choosing an appropriate output from redundant components. The concrete application of this idea would, for example, be the Triple Modular Redundancy pattern which uses the *Voting* tactic to choose for the majority of three redundant subsystem outputs.

It is difficult to keep tactics and patterns apart as there is no clear boarder between the two. For example, (Saridakis, 2002) describes different forms of degradation as fault tolerances patterns, whereas (Wu, 2003) considers *Degradation* as a safety tactic. (Ryoo et al., 2010) specify some criteria to identify tactics. For a design decision on order to be a tactic, it has to be atomic. This means that it cannot be divided into other multiple tactics, however it can be refined. For example, the *Redundancy* tactic is refined by the *Replication Redundancy* tactic and the *Diverse Redundancy* tactic, but it is not composed of them. Furthermore, (Ryoo et al., 2010) say that tactics focus on a single quality attribute (e.g. safety) and patterns usually affect several quality attributes.

There are also different opinions whether tactics are basic building blocks for design patterns (as in (Kumar and Prabhakar, 2010a)) or whether tactics

[2] The 2oo4 architecture is a special version of the M-out-of-N Pattern which is explained on page 28.

are used as additional design enhancements for certain design patterns (as in (Harrison and Avgeriou, 2008)).

In this article we take the list of safety tactics from (Preschern et al., 2013c) and we consider safety tactics as building blocks for safety architecture patterns. The full list of safety tactics is given in Appendix B and shows the tactic names, a short description of the tactics, and the methods of the IEC 61508 safety standard which can be related to these tactics.

3.2 Goal Structuring Notation

The Goal Structuring Notation (GSN) was developed by (Kelly and Weaver, 2004) and is often used in the safety domain to provide a structured argument for the achievement of specific goals. Recently, a standard for GSN was published which contains definitions of the notation and which presents approaches how to use GSN to elaborate a specific goal (Spriggs, 2012). GSN can also be used to argue for system security like, for example, in (Cockram and Lautieri, 2007). We will use GSN to argue for safety and for security of the presented patterns and we will use the GSN concepts from Figure 1.

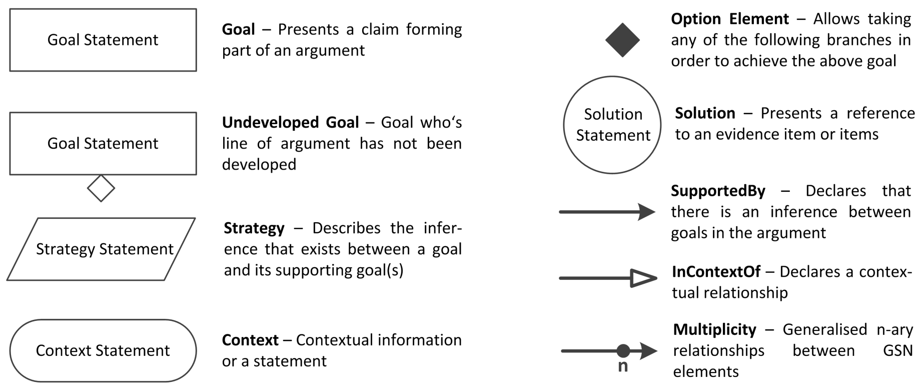

Goal Statement
Goal – Presents a claim forming part of an argument
Goal Statement
Undeveloped Goal – Goal who's line of argument has not been developed
Strategy Statement
Strategy – Describes the inference that exists between a goal and its supporting goal(s)
Context Statement
Context – Contextual information or a statement
Option Element – Allows taking any of the following branches in order to achieve the above goal
Solution Statement
Solution – Presents a reference to an evidence item or items
SupportedBy – Declares that there is an inference between goals in the argument
InContextOf – Declares a contextual relationship
n
Multiplicity – Generalised n-ary relationships between GSN elements

Fig. 1. GSN concepts used in this article, based on (GSN Working Group, 2011)

To show how a GSN goal is achieved, it is linked to an argument (GSN strategies, GSN subgoals) which ends up in the evidence (GSN solutions) supporting the claim that the goal is achieved. Figure 2 shows an example for the application of GSN. The main goal in the example is that an attacker cannot obtain some confidential data. In the next step, context elements are added which say that the data is locally stored on a computer and transmitted to another computer. The main goal is split up into the subgoals to protect the stored data and the transmitted data. Protecting transmitted data is achieved by transmitting data over a protected TLS channel (GSN strategy). For this TLS channel, we need evidence that it is reliable. This evidence (GSN solution element) is that the used implementation is robust and proven in use. Protecting stored data is an undeveloped goal which means that the security argument for this subgoal is not yet complete and further arguments have to be included here in order to obtain a complete argument that the overall goal to protect data is achieved. In this example, GSN provides a structured way to show how the rather unspecific goal to protect confidential data is (partially) achieved by specific measures (the TLS channel).

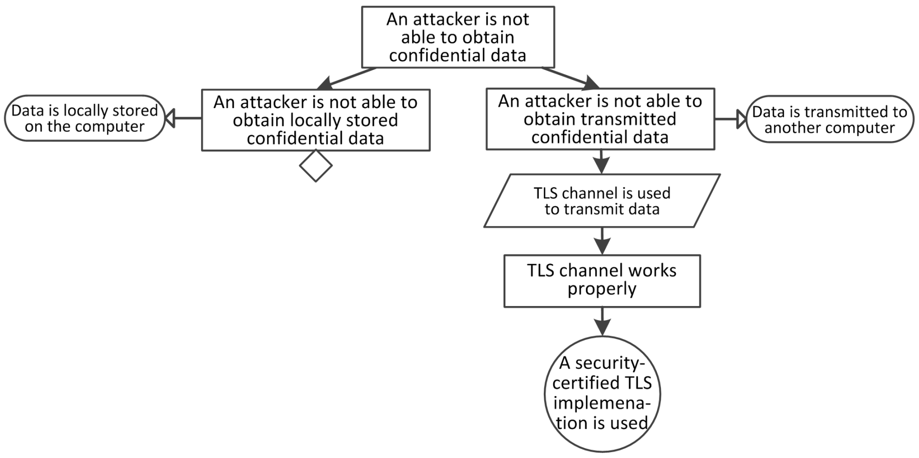

An attacker is not able to obtain confidential data
An attacker is not able to obtain locally stored confidential data
Data is locally stored on the computer
An attacker is not able to obtain transmitted confidential data
Data is transmitted to another computer
TLS channel is used to transmit data
TLS channel works properly
A security-certified TLS implemena-tion is used

Fig. 2. GSN example showing a security argument

3.3 STRIDE Analysis

In order to build a secure system, it is necessary to first find the relevant threats to the system before finding solutions how to mitigate them. The STRIDE approach is a structured way to find these threats. The STRIDE approach was proposed by Microsoft (Howard and LeBlanc, 2003) and is nowadays often used for security engineering. STRIDE is an acronym, where the letters stand for the six threat categories which are analyzed (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), Elevation of Priviledge (EoP)).

For threat modeling with STRIDE, first a data flow diagram has to be constructed. A data flow diagram shows the interaction between system elements and external elements (e.g. users of the system) by graphically presenting all the data flows (inputs/outputs of elements). All relevant STRIDE threats for each element in the diagram are then listed. The relevant threats for different data flow diagram elements types are given in Table 2.

The resulting list of threats can further be elaborated by excluding threats which are not relevant for the specific system and by implementing countermeasures for relevant threats. When all threats are covered, one has a structured argument for system security. We use Goal Structuring Notation to present such a structured argument.

data flow diagram element type	**S**	**T**	**R**	**I**	**D**	**E**
External entity	X		X			
Data flow		X		X	X	
Data store		X	X	X	X	
Process	X	X	X	X	X	X

Table 2. STRIDE mapping to data flow diagram element types

4 Safety Pattern System

This section explains with an example (the Homogenous Duplex Pattern) how to include a safety architecture pattern into our pattern system. We explain the applied pattern format and we describe how to bring a safety pattern into this format (which includes building GSN diagrams for safety and security). Furthermore, we describe how to find the relationships to other patterns. An overview of this described approach is shown in Figure 3. This approach was applied for all the safety patterns presented in Appendix A.

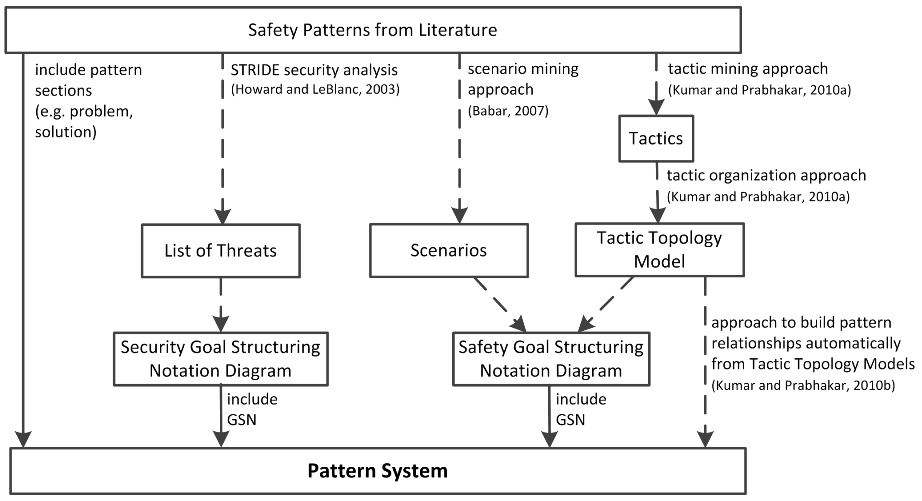

Safety Patterns from Literature
include pattern sections (e.g. problem, solution)
STRIDE security analysis (Howard and LeBlanc, 2003)
scenario mining approach (Babar, 2007)
tactic mining approach (Kumar and Prabhakar, 2010a)
Tactics
tactic organization approach (Kumar and Prabhakar, 2010a)
List of Threats
Scenarios
Tactic Topology Model
approach to build pattern relationships automatically from Tactic Topology Models (Kumar and Prabhakar, 2010b)
Security Goal Structuring Notation Diagram
Safety Goal Structuring Notation Diagram
include GSN
include GSN
Pattern System

Fig. 3. Applied approach to build the pattern system

4.1 Pattern Format

We use the pattern format presented by (Babar, 2007) for all our safety architecture patterns. The pattern format explicitly provides architectural information like relevant scenarios with the aim to aid architecture design and evaluation processes. Table 3 shows which sections the pattern format contains and where we got the information for these sections from.

Section	What it contains and where the contents comes from
Pattern Name	The pattern name is taken from the existing pattern - most of which come from (Armoush, 2010).
Pattern Type	Classification into hardware/software and fail-safe/fail-over. We classify these pattern types during the process of building up the pattern language.
Also Known As	Other names for the pattern used in literature.
Context	The contents of this section comes from existing patterns and was structurally adapted to fit our pattern system.
Problem	The contents of this section comes from existing patterns and was structurally adapted to fit our pattern system.
Forces	The contents of this section comes from existing patterns (mostly from (Grunske, 2003)) and was structurally adapted to fit our pattern system.
Solution	The solution is shortly described in a few sentences and the structure of the safety architecture is shown in a diagram. Most of the diagrams are based on (Armoush, 2010) and (Douglass, 2002).
Safety GSN	This section contains a Goal Structuring Notation (GSN) diagram which relates the main safety aim of the pattern to the architectural design decisions which were taken to achieve this aim. The GSN diagram is based on information about the usage of basic architectural design decisions (safety tactics) which are applied in the pattern.
Security GSN	This section contains a Goal Structuring Notation (GSN) diagram with the main goal to maintain safety also in case of attacks. The STRIDE method is used to identify threats for the pattern and the GSN diagram structurally presents relevant threats.
Consequences	The consequences are split into a part containing general consequences and a part explicitly covering quality-attribute related consequences (e.g. consequences on safety or security). Information about consequences mostly comes from the safety patterns from (Armoush, 2010) and (Grunske, 2003).
General Scenarios	This section contains scenarios of the system which can, for example, be used during architecture evaluations. The information of the scenarios is also needed to build the safety GSN diagrams.
Known Uses	This section presents known uses for the patterns. We added this information by searching for literature which applies the pattern. We just included patterns for which we could find at least three known uses.
Credits	References to previous work on the pattern.

Table 3: Applied pattern format

4.2 Elaborating Pattern Sections

The following pattern sections of the presented safety patterns are based on safety patterns from literature: **Pattern Name**, **Also Known As**, **Context**, **Problem**, **Forces**, **Solution**, **Consequences**. Most of the information is mainly based on (Armoush, 2010) and is further elaborated by using other literature on similar safety patterns. This means we consider several patterns from literature, gather all information, and bring it into our format to be consistent.

For example, (Armoush, 2010) describes the Homogenous Redundancy Pattern, which is also described by (Douglass, 2002) and (Grunske, 2003). We take the pattern from (Armoush, 2010) as a starting point and include more detailed information about the **Solution** from (Douglass, 2002) and about the **Forces** from (Grunske, 2003).

We added the **Known Uses** section to the safety patterns by searching for examples in literature which apply the patterns. We just included safety patterns for which we could at least find three known uses. We also added the **Credits** section to state where we obtained the pattern information from.

Section 4.3 described how we constructed the **Safety GSN** section and how we obtained **General Scenarios** for the patterns. Section 4.5 shows how we construct the **Security GSN**.

4.3 Developing the Safety GSN

In this section we construct the safety GSN diagram for the HOMOGENOUS DUPLEX PATTERN based on safety tactics applied by this pattern.

Mining Patterns for Safety Tactics

The HOMOGENOUS DUPLEX PATTERN is mentioned in literature by (Douglass, 2002), (Grunske, 2003), and (Armoush, 2010). We studied all three descriptions of the pattern to find text passages which indicate the usage of general safety-related architectural design decisions (safety tactics). We do this as proposed by (Kumar and Prabhakar, 2010a), where tactics are mined from GoF and POSA patterns to find relationships between patterns which use similar tactics. We apply the same method to find relationships between safety architecture patterns.

As proposed by (Kumar and Prabhakar, 2010a), we construct a table of pattern text passages and corresponding tactics that this text passage relates to. For example, the HOMOGENOUS DUPLEX PATTERN pattern in (Armoush, 2010) says: "*The system consists of two identical modules*" This indicates that the pattern applies the *Replication Redundancy* safety tactic[3].

Table 4 shows the table with the pattern text passages and also shows the tactics which we found in the HOMOGENOUS DUPLEX PATTERN. The tactics are: *Replication Redundancy*, *Override*, and *Condition Monitoring.*

Abstract Section		
Core Intent		**Tactic**
It is a hardware pattern that is used to increase the safety and reliability of the system by providing a replication of the same module (Modular redundancy).		Replication Redundancy
Problem Section		
Problem	**Elaboration of Problem (scenario)**	**Tactic**
Make the system continue operating in the presence of a fault	The system is fully operational even in case of a single channel failure. A single channel random fault does not lead to a system failure. The system can detect a fault in a single channel.	Replication Redundancy Override
Solution Section		
Solution Description		**Tactic**
The system consists of two identical modules; a primary (active) module and secondary (standby). *Test by redundant hardware*		Replication Redundancy
There is a fault detection unit that monitors the primary module and switches to the secondary module when a fault appears in the primary. *Fault detection and diagnosis (Comparator and Acceptance Test)*		Override
This method performs a check on the two channels by checking for input valid data within a given range and by checking the output signals from the two modules.		Condition Monitoring
Consequences Section		
Consequence Description		**Tactic**
When a fault is detected in the primary channel, the switch circuit switches over to the secondary channel		Override
Implementation Section		
Implementation Description		**Tactic**
To implement this pattern, the computational channel should be duplicated		Replication Redundancy

Table 4. Mining Tactics of the HOMOGENOUS DUPLEX PATTERN

[3] A list of all safety tactics is available in Appendix B

Building a Structured Tactic Representation

With the gathered tactics we construct a *Tactic Topology Model* with is also part of the method described by (Kumar and Prabhakar, 2010a). First, one has to think about the main goal of the pattern. According to (Kumar and Prabhakar, 2010a), the main tactics which achieve this goal are usually related to the **Intent** or the **Problem** section of the pattern. In the Tactic Topology Model, these main tactics are connected to the patterns' goal with arrows. Further explanation about this connection is given in textual form next to the arrow. The tactics can bring up new goals which have to be achieved by additional tactics - these are also added with arrows and a textual description. In that way, a structured graph containing the patterns' tactics is constructed.

Figure 4 shows the Tactic Topology Model for the Homogenous Redundancy Pattern. The main goal in the Tactic Topology Model for the pattern is identified as *"Continue operation even in case of faults"*. The two tactics directly connected to this goal are *Replication Redundancy* and *Override*, because they are mentioned in the **Core Intent** or **Problem** section of the pattern (see Table 4). An additional goal that has to be fulfilled is to detect when the *Override* tactic should switch to the backup channel. Therefore, the *Condition Monitoring* tactic is connected to the *Override* tactic.

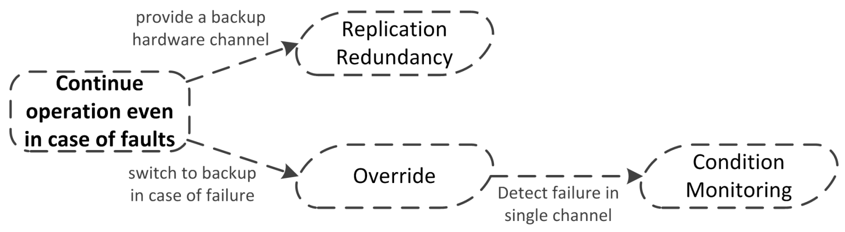

provide a backup hardware channel
Replication Redundancy
Continue operation even in case of faults
switch to backup in case of failure
Override
Detect failure in single channel
Condition Monitoring

Fig. 4. Tactic Topology Model for the Homogenous Duplex Pattern

We use the Tactic Topology Models to structurally establish relationships in our pattern system (this is explained in Section 4.4). Apart from that, the Tactic Topology Models are just intermediate results used to build GSN diagrams and are not included in the patterns. All Tactic Topology Models for the presented safety patterns can be found in (Preschern et al., 2013b).

Constructing the Safety GSN

Based on the Tactic Topology Model, we construct the safety GSN diagram. The GSN diagram contains the tactics from the Tactic Topology Model and they additionally contain general scenarios which are mined from the pattern descriptions. This scenario mining is done as proposed in (Babar, 2007) by searching the **Problem** and **Solution** statements for safety-related scenarios. The scenarios found for the Homogenous Duplex Pattern are shown in Table 4 under **"Problem Section"**.

All our safety GSNs start with the main goal to maintain system safety. This main goal is split up into subgoals based on the scenarios which we obtained from the patterns. If the scenarios are independent from each other, then they are put on the same level in the GSN. If a scenario depends on another scenario (as it is the case for the HOMOGENOUS DUPLEX PATTERN), then it is modeled as a subgoal of the scenario it depends on. The tactics which are necessary to achieve a GSN goal are put below this (sub-)goal as a GSN strategy which has the title of the tactic and which contains additional information (taken from the textual description of the Tactic Topology Model arrow connections). GSN context elements are added if information of the pattern's context section is relevant for the GSN goals.

Figure 5 shows the GSN diagram of the HOMOGENOUS DUPLEX PATTERN pattern. We can see that it consists of all the tactics of the Tactic Topology Model from Figure 4 and of the scenarios of the HOMOGENOUS DUPLEX PATTERN from Table 4.

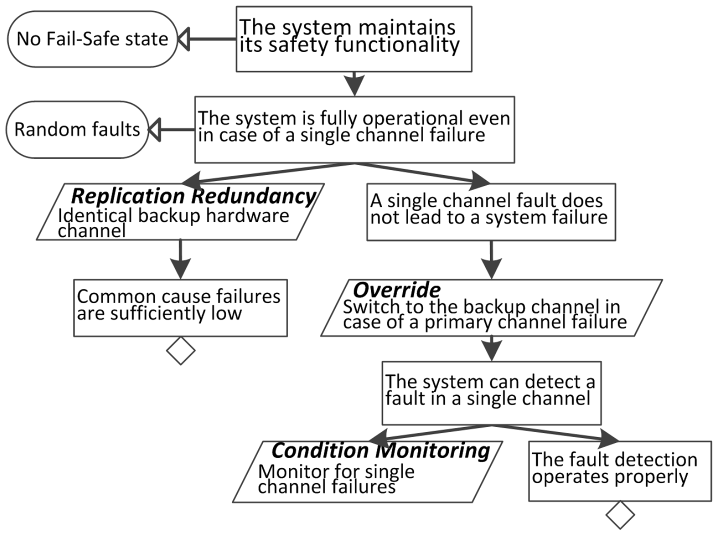

No Fail-Safe state
The system maintains its safety functionality
Random faults
The system is fully operational even in case of a single channel failure
Replication Redundancy Identical backup hardware channel
A single channel fault does not lead to a system failure
Common cause failures are sufficiently low
Override Switch to the backup channel in case of a primary channel failure
The system can detect a fault in a single channel
Condition Monitoring Monitor for single channel failures
The fault detection operates properly

Fig. 5. GSN diagram of the HOMOGENOUS DUPLEX PATTERN

4.4 Finding Pattern Relationships

To obtain the relationships between the patterns, we use the approach presented by (Kumar and Prabhakar, 2010b). They compare Tactic Topology Models of patterns and define a mapping between Tactic Topology Model predicates and pattern relationships. For example, if the Tactic Topology Models of two different patterns are equal, then (Kumar and Prabhakar, 2010b) say that these patterns are similar. Table 4.4 shows all kinds of considered pattern relationships. To find all relationships in a pattern system, every patterns' Tactic Topology Model has to be compared to the Tactic Topology Models of all other patterns and every such Tactic Topology Model pair has to be checked for all the predicates described in Table 4.4.

Relationship	**Description**	**Tactic Topology Model predicate**
is an alternative	Patterns A and B solve the same problem, but propose different solutions.	$SourceNode(A) = SourceNode(B)$ **AND** $Graph(A) \neq Graph(B)$
uses	A sub-problem of pattern A is similar to the problem addressed by pattern B.	$Graph(A) \supset Graph(B)$
refines	Pattern B provides a more detailed solution than pattern A.	$SourceNode(A) = SourceNode(B)$ **AND** $Graph(A) \subset Graph(B)$
specializes	The solution of pattern B is a special case of the solution of pattern A. *Example: Pattern B specializes pattern A if they have the same graph structure, but pattern B uses a refined tactic where pattern A uses a more general tactic (e.g. B uses Replication Redundancy where A uses Redundancy).*	$Graph(A) \subset generalizedGraph(B)$
is similar	Patterns A and B provide the same solution to a similar problem *Example: Pattern B is similar to pattern A if they have the same graph structure and use two related refined tactics. E.g. A uses Replication Redundancy and B uses Diverse Redundancy*	$generalizedGraph(A) \equiv generalizedGraph(B)$

Table 5. Description of pattern relationships (slightly modified from (Kumar and Prabhakar, 2010b))

We applied this approach to our safety patterns to structurally build the relationships in our pattern system. We built the Tactic Topology Models as described in Section 4.3 for all our patterns. Then we compared each Tactic Topology Models with one another and checked for the predicates defined in Table 4.4. This delivers us the pattern relationships for our pattern system.

Figure 6 shows our safety patterns and their relationships which we obtained with the described approach. We can see that the approach to find pattern relationships worked out quite well. All the relationships between the patterns seem to be comprehensible. For example, according to the relationships obtained through the Tactic Topology Model comparison, the Homogenous Duplex Pattern is a specialization of the M-out-of-N-D Pattern and is similar to the Heterogenous Duplex Pattern which is both reasonable.

To not overload the the pattern-relationship representation, we did not explicitly annotate the *is alternative* relationships, but instead grouped patterns which are alternatives to one another into the group of patterns trying to maintain a safe-state in case of faults and the group of patterns providing full system functionality in case of faults. Additionally, we divided the patterns into software and hardware patterns as already suggested by (Armoush, 2010). However, the classification of software and hardware patterns is not very strict. Some of the patterns are intended for either software or hardware, but could also be implemented for the other. For example, the Watchdog pattern is a hardware pattern, but could also be realized in software by a timer which watches the execution of another program. The group a pattern belongs to is stated in the **Pattern Type** section. All patterns from Figure 6 are presented in Appendix A.

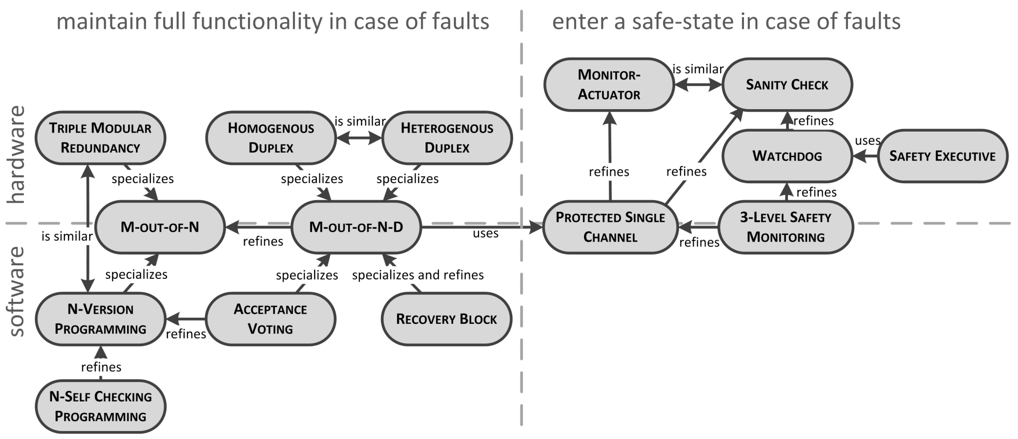


Fig. 6. Safety Architecture Pattern System

4.5 Developing the Security GSN

In this section we apply the STRIDE analysis to the Homogenous Duplex Pattern to obtain a GSN security representation. More detailed information about this approach can be found in (Preschern et al., 2013d).

Getting the Data Flow Diagram
The covered safety architecture patterns each provide a diagram which shows the hardware and software elements of the pattern and their interaction. This diagram can be used as a data flow diagram in the STRIDE analysis.

Figure 7 shows such a diagram for the Homogenous Duplex Pattern. The system gets input data and processes that input data with redundant channels which provide output data. A fault-detector and a switch component decide which of the channel outputs is used.

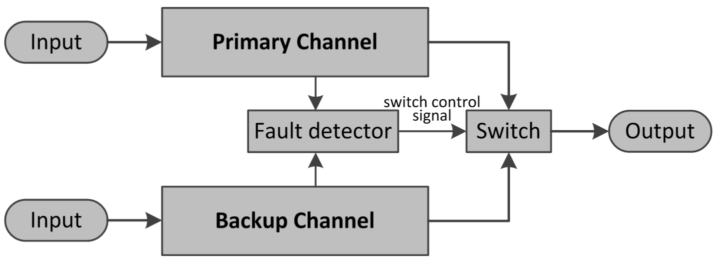


Fig. 7. Homogenous Duplex Pattern architecture

Applying the STRIDE Analysis
By using an adapted STRIDE approach, we analyze the pattern diagrams to list the security threats for each of the patterns.

We just consider two element types for the STRIDE analysis: *Data Flows* and *Processing Elements*. For both types, we omit the threats Repudiation and Information Disclosure, because they do not directly influence the safety functionality of a system. Furthermore, for the *Processing Elements*, we omit the Tampering and Denial of Service threats, because an attacker usually has no access to processing elements which perform safety-critical functionality. Therefore, he needs to elevate his privileges before starting a tampering or DoS attack on a processing element. Our resulting relevant threats for the pattern diagram element types are shown in Table 6.

DFD element type	**Symbol**	**S**	**T**	**R**	**I**	**D**	**E**
Data Flow	→		X			X	
Processing Element	▭	X					X

Table 6. STRIDE mapping to safety pattern element types

With this mapping of relevant threats, we go through each element of the Homogenous Duplex Pattern to obtain a list of security threats. We did not consider the identical *Primary Channel* and *Secondary Channel* separately, but we just cover a *Single Channel* which can be either of them. For the Homogenous Duplex Pattern we get the following list of threats:

– Tampering of *Single Channel* input data
– DoS of *Single Channel* input data
– Spoofing of *Single Channel*
– EoP on *Single Channel*
– Tampering of *Single Channel* output data
– DoS of *Single Channel* output data
– Tampering of *Fault Detector* input data
– DoS of *Fault Detector* input data
– Spoofing of *Fault Detector*
– EoP on *Fault Detector*
– Tampering of *Fault Detector* output data
– DoS of *Fault Detector* output data
– Spoofing of *Switch*
– EoP on *Switch*
– Tampering of *Switch* output data
– DoS of *Switch* output data

From this list of threats it is not very easy to grasp important threats, because some of the listed threats do not even pose a direct threat to system safety.

For example, the *DoS of Fault Detector output data* threat itself cannot bring the system into a safety-critical state, because as long as both channels work properly, it does not matter which channel output is selected. Therefore, this threat is not as relevant as for example the *DoS of Switch output data* threat which can bring the system into a safety-critical state in any case.

Constructing the Security GSN
Based on the list of threats for a pattern, we develop a GSN diagram to argue that attacks cannot affect system safety. The security GSN diagram for the Homogenous Duplex Pattern is shown in Figure 8.

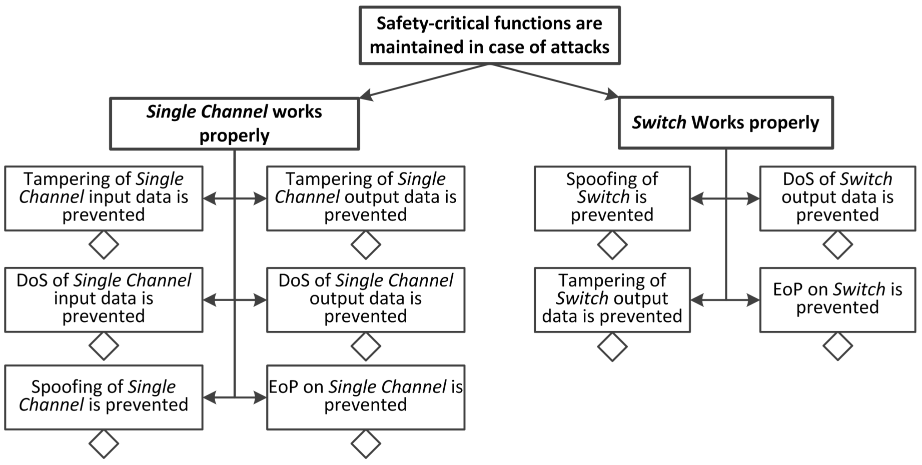

Safety-critical functions are maintained in case of attacks
Single Channel works properly
Switch Works properly
Tampering of *Single Channel* input data is prevented
Tampering of *Single Channel* output data is prevented
DoS of *Single Channel* input data is prevented
DoS of *Single Channel* output data is prevented
Spoofing of *Single Channel* is prevented
EoP on *Single Channel* is prevented
Spoofing of *Switch* is prevented
DoS of *Switch* output data is prevented
Tampering of *Switch* output data is prevented
EoP on *Switch* is prevented

Fig. 8. Homogenous Duplex Pattern Security GSN

Each security GSN starts with the main aim that “Safety-critical functions are maintained in case of attacks”. Next, all threats are added which can bring the system into a safety-critical state. Threats are represented as undeveloped GSN goals in form of: “Threat X is prevented”. If the pattern is applied, these undeveloped goals have to be further developed to obtain a complete security argument (evidence that the goals are achieved has to be provided).

An example for a threat in the GSN diagram of the Homogenous Duplex Pattern is “Tampering of Switch output data”. If someone mounts an attack related to this threat, the attacker can produce arbitrary system output data which violates system safety. Therefore, the threat is safety-critical and is directly added to the GSN diagram as “Tampering of Switch output data is prevented” GSN goal[4].

[4] Actually the threat is added after the “Switch works properly” GSN goal. This goal is just introduced to make the GSN diagram easier to read and it changes nothing about the semantics of the diagram

After all safety-critical threats are included in the GSN diagram, all the remaining threats (which by themselves cannot bring the system into a safety-critical state) are considered. Any combination of these threats is also added to the GSN diagram if it can affect system safety (combinations are notated in the diagram with the *GSN Option Element*, however none are present for the Homogenous Duplex Pattern).

The remaining threats are not directly relevant for the safety functionality of the system and are therefore not included in the GSN diagram. For example, in the Homogenous Duplex Pattern there are threats related to the Fault Detector unit. None of the threats related to the Fault Detector are included in the GSN diagram, because if an attacker has full control of the Fault Detector, he can just influence which of the two channels is actually used for the system output. However, if both channels work properly this is not safety-critical. Still, with an attack on the Fault Detector, an attacker can disable the systems' safety functionality, because switching to the redundant channel in case of failure would not work anymore. Still, we do not consider such threats, because they are much less critical than the ones included in the GSN.

5 Pattern Application - Case Study

In this section we describe the application of the Homogenous Duplex Pattern in an industrial case study. We apply the pattern and use the safety and security GSN diagrams to argue for system safety and security. An overview of this process is shown in Figure 9.

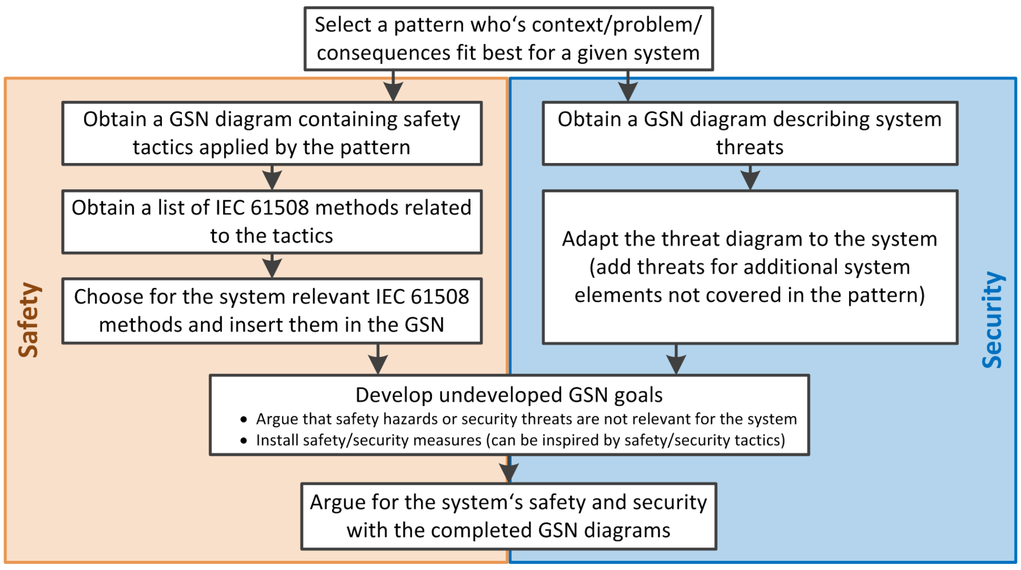

Select a pattern who's context/problem/ consequences fit best for a given system
Obtain a GSN diagram containing safety tactics applied by the pattern
Obtain a list of IEC 61508 methods related to the tactics
Choose for the system relevant IEC 61508 methods and insert them in the GSN
Obtain a GSN diagram describing system threats
Adapt the threat diagram to the system (add threats for additional system elements not covered in the pattern)
Develop undeveloped GSN goals
• Argue that safety hazards or security threats are not relevant for the system
• Install safety/security measures (can be inspired by safety/security tactics)
Argue for the system's safety and security with the completed GSN diagrams
Safety
Security

Fig. 9. Building a safety/security argument based on a pattern

5.1 System Description

A controller for a hydro-power plant is developed. The controller unit has to maintain safety-critical functions such as controlling the turbine speed and shutting down the system if an overvoltage is detected. Failing to maintain these functions can result in damage of the equipment and even human injuries. Sensors are connected to an IEC 61850 merging unit which is a separate controller collecting sensor data and transmitting this data to the hydro-power plant controller via Ethernet. Based on the sensor input, the hydro-power plant controller has to control actuators which are hardwired to the controller. The controller is connected to the local hydro-power plant network to be able to receive firmware updates. However, this is a different Ethernet interface than the one used by the merging unit.

For this system, the Homogenous Duplex Pattern (see Appendix A on page 24) is applied to protect from hardware failures. Figure 10 shows the resulting architecture. The system consists of two main CPUs which are supplied with their own set of sensor data from different merging units. Both CPUs run the same software to compute outputs for the actuators. A separate switch component determines which output is actually used for the actuators. The switch usually takes CPU1, however, if the watchdog component which periodically sends challenges to both CPUs detects that one CPU does not work, it instructs the switch to take the output of the other CPU.

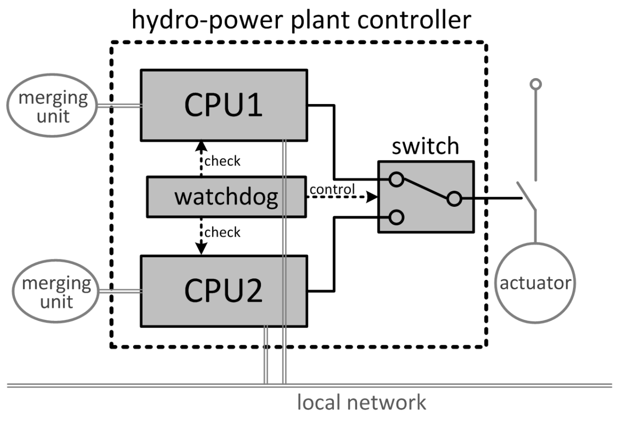


Fig. 10. Hydro-power plant controller architecture

5.2 Safety Argument

From the Homogenous Duplex Pattern we obtain the basis for the safety GSN diagram which was already shown in Figure 5 on page 11. From this diagram we can see that the pattern applies the *Replication Redundancy*, the *Override*, and the *Condition Monitoring* tactics. Table 7 shows the related IEC 61508 methods for these tactics (taken from Appendix B). Thus, when applying a pattern, one gets a list of relevant IEC 61508 conform methods for the specific architecture. From this list, a safety architect now has to choose which of the suggested IEC 61508 methods are appropriate for the specific system.

Tactics	Related IEC 61508 methods	
Replication Redundancy	A.2.1 Test by redundant hardware	✓
	A.2.5 Monitored redundancy	
	A.3.5 Reciprocal comparison by software	
	A.4.5 Block replication	
	A.6.3 Multi-channel output	
	A.6.5 Input comparison/Voting	
	A.7.3 Complete hardware redundancy	✓
	A.7.5 Transmission redundancy	
Override	A.1.3 Comparator	✓
	A.1.5 Idle current principle	
	A.8.1 Overvoltage protection with safety shut-off	
	A.8.3 Power-down with safety shut-off	
Condition Monitoring	A.1.1 Failure detection by online monitoring	✓
	A.6.4 Monitored output	
	A.8.2 Voltage control	
	A.9.1 Watch dog with separate time base without time-window	
	A.9.2 Watch dog with separate time base and time-window	
	A.9.3 Logical monitoring of program sequence	
	A.9.4 Temporal and logical program sequence monitoring	
	A.9.5 Temporal monitoring with on-line check	✓
	A.12.1 Reference sensor	
	A.13.1 Monitoring	

Table 7. Tactics and IEC 61508 methods used by the Homogenous Duplex Pattern

In our case study, the methods from Table 7 which are marked with a tick were chosen. These methods are then included in the GSN diagram instead of the general tactics. Furthermore, additional information can be added to the GSN diagram if the safety argument is not complete. For example, all undeveloped goals have to be developed which means that they have to be linked to evidence which suggests that the goal is fulfilled. Figure 11 shows the resulting GSN diagram (additionally added elements are presented in orange, dashed lines). The GSN shows how the high level safety goal can be achieved by the actually applies methods suggested by the IEC 61508 safety standard.

5.3 Security Argument

To develop a security argument, we start with the security GSN diagram provided by the pattern and complete all the undeveloped goals. For all the undeveloped goals we either argue why the goal is not relevant or what measures are applied to handle the threat. An example for not relevant threats are any threats to the switch element, because the switch element for the hydro-power plant architecture is hardwired to the other elements and not accessible for an attacker. An example for measures against a threat is related to the *"EoP on Single Channel is prevented"* goal. Do detect some attacks related to this threat, a runtime-integrity-checker is used to detect malware on the CPUs. Figure 12

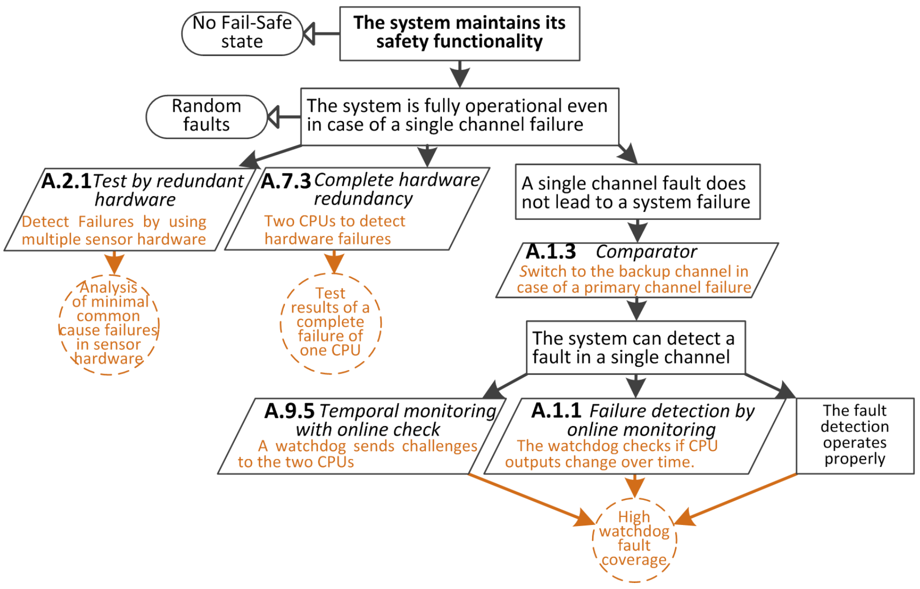

No Fail-Safe state
The system maintains its safety functionality
Random faults
The system is fully operational even in case of a single channel failure
A.2.1 Test by redundant hardware
Detect Failures by using multiple sensor hardware
A.7.3 Complete hardware redundancy
Two CPUs to detect hardware failures
A single channel fault does not lead to a system failure
A.1.3 Comparator
Switch to the backup channel in case of a primary channel failure
Analysis of minimal common cause failures in sensor hardware
Test results of a complete failure of one CPU
The system can detect a fault in a single channel
A.9.5 Temporal monitoring with online check
A watchdog sends challenges to the two CPUs
A.1.1 Failure detection by online monitoring
The watchdog checks if CPU outputs change over time.
The fault detection operates properly
High watchdog fault coverage

Fig. 11. Hydro-power plant controller safety GSN diagram

shows the complete security GSN diagram for the hydro-power plant architecture. Additionally added elements in the GSN are marked with blue, dashed lines. This GSN shows how the high level goal that the safety functionality cannot be influenced by attacks is achieved by specific measures and arguments.

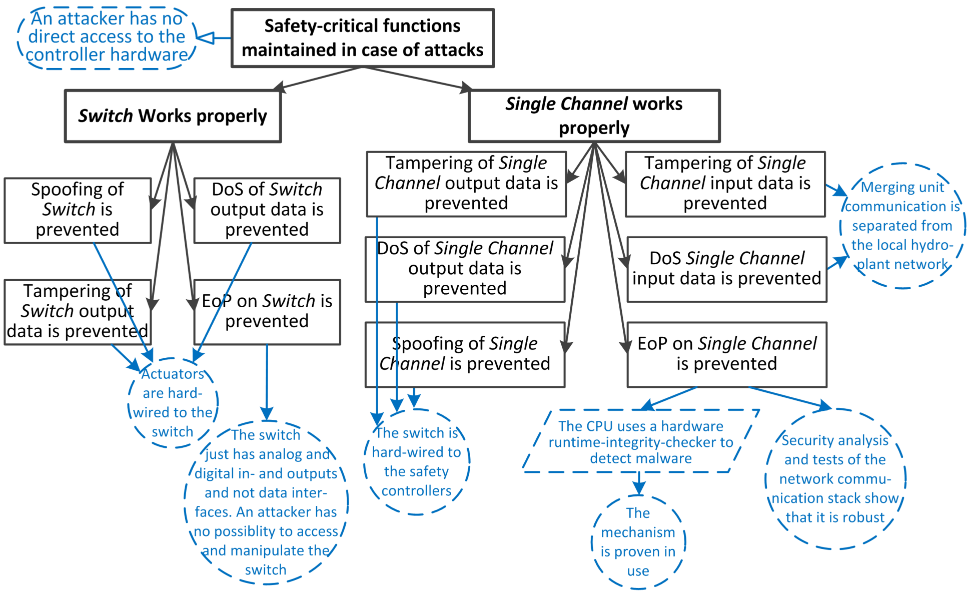

An attacker has no direct access to the controller hardware
Safety-critical functions maintained in case of attacks
Switch Works properly
Single Channel works properly
Spoofing of Switch is prevented
DoS of Switch output data is prevented
Tampering of Switch output data is prevented
EoP on Switch is prevented
Tampering of Single Channel output data is prevented
Tampering of Single Channel input data is prevented
DoS of Single Channel output data is prevented
DoS Single Channel input data is prevented
Spoofing of Single Channel is prevented
EoP on Single Channel is prevented
Merging unit communication is separated from the local hydro-plant network
Actuators are hard-wired to the switch
The switch just has analog and digital in- and outputs and not data interfaces. An attacker has no possiblity to access and manipulate the switch
The switch is hard-wired to the safety controllers
The CPU uses a hardware runtime-integrity-checker to detect malware
The mechanism is proven in use
Security analysis and tests of the network communication stack show that it is robust

Fig. 12. Hydro-power plant controller security GSN diagram

5.4 Discussion

With the application of the safety pattern we get a blueprint for a safety and for a security GSN diagram.

– For safety, system architects get the benefit, that they are provided with suggestions for IEC 61508 methods. Additionally the safety GSN provides a way to argue how the main goal to keep the system safe can be achieved by implementing the IEC 61508 methods. (Preschern et al., 2013a) evaluates the quality of IEC 61508 method suggestion by comparing the pattern suggestions to real-life safety projects.
– For security, system architects get the benefit that on the one hand they are introduced to a security analysis technique and on the other hand that they are already provided with a security GSN diagram for the pattern which they can use and extend to argue for system security. To implement security methods in order to mitigate the threats in the GSN diagram, (Preschern et al., 2012b) provides a list of general methods in form of security tactics.

For both, the safety and security GSN, it is always difficult to say whether the arguments provided are enough to argue that the main goal is achieved.

– For the safety GSN we use relevant scenarios from the patterns. (Wu, 2007) argues that if a GSN contains all relevant scenarios for the main goal, the argument is sufficiently complete.
– For the security GSN we use STRIDE to include all relevant threats in the GSN.

However, to make sure that the constructed GSNs include all safety and security aspects of an architecture, other relevant scenarios could be included into the safety GSN and additional elements of the architecture which are not included in the patterns have to be analyzed with STRIDE for the security GSN. Thus, the provided GSNs can be used as a basis for safety and security reasoning, but they have to be adapted or extended for specific architectures, especially, if compared to the pattern additional elements are present.

6 Conclusion

This article presented a system of safety architecture patterns which contain a security analysis. A detailed explanation how these patterns can be constructed was given and a case study applying one of the patterns was shown.

The provided patterns can be used by system architects to develop safety-critical architectures and to additionally see how to apply security analysis methods. This has the benefit that the patterns provide a starting point how to analyze and argue for system security. From a safety point of view the patterns bring the advantage that they provide a way to reason about system safety. They additionally provide suggestion for IEC 61508 methods as a starting point for safety architects to see which methods of the safety standard are relevant for the chosen architecture.

For future work, further patterns could be included to the pattern system. It would be interesting to see if the approach to find pattern relationships with Tactic Topology Models also works for other safety patterns. As well it would be interesting to see if the approach to find pattern relationships can also be applied to other domains such as security patterns. Furthermore, it would be interesting to see the safety patterns including the safety and security GSNs applied in other real-life projects to be able to better discuss the benefits and shortcomings of the patterns.

With the presented patterns we want to give safety architects guidance for efficiently constructing well-proven architectures and we hope to increase the security awareness in the safety domain.

References

Antonino, Pablo Oliveira, Thorsten Keuler, and Pablo Antonino (2012). Towards an Approach to Represent Safety Patterns. In: *The Seventh International Conference on Software Engineering Advances (ICSEA)*. c, 228–237.

Armoush, Ashraf (2010). Design patterns for safety-critical embedded systems. PhD thesis. RWTH Aachen University.

Babar, Muhammad Ali (2007). Improving the Reuse of Pattern-Based Knowledge in Software Architecting. In: *EuroPLoP*. Lero, Ireland, 7–11.

Bachmann, Felix, Len Bass, and Mark Klein (2003). *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design*. Tech. rep. March. Carnegie Mellon Software Engineering Institute.

Buschmann, Frank et al. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons.

Cockram, T.J. and S.R. Lautieri (2007). Combining Security and Safety Principle in Practice. In: *2nd Institution of Engineering and Technology International Conference on System Safety*. IEEE, 159–164.

Daniels, Fonda, Kalhee Kim, and Mladen A Vouk (1997). The Reliable Hybrid Pattern A Generalized Software Fault Tolerant Design Pattern. In: *European Conference on Pattern Language of Programs (EuroPLoP)*, 1–9.

Douglass, Bruce Powel (1998). Safety-Critical Systems Design. *Electronic Engineering* 70, 862.

Douglass, Bruce Powel (2002). *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Pearson.

Douglass, Bruce Powel (2010). *Design Patterns for Embedded Systems in C*. Elsevier.

Gawand, Hemangi, RS Mundada, and P. Swaminathan (2011). Design Patterns to Implement Safety and Fault Tolerance. *International Journal of Computer Applications* 18, 2, 6–13.

Grunske, Lars (2003). Transformational Patterns for the Improvement of Safety Properties in Architectural Specification. In: *Proceedings of The Second Nordic Conference on Pattern Languages of Programs (VikingPLoP)*.

GSN Working Group (2011). *GSN Community Standard Version 1*. http://www.goalstructuringnotation.info/.

Halkidis, S., A. Chatzigeorgiou, and G. Stephanides (2004). A qualitative evaluation of security patterns. In: *6th International Conference on Information and Communications Security*. Springer, 132–144.

Halkidis, S., A. Chatzigeorgiou, and G. Stephanides (July 2006a). A qualitative analysis of software security patterns. *Computers & Security* 25, 5, 379–392.

Halkidis, S., A. Chatzigeorgiou, and G. Stephanides (2006b). Quantitative Evaluation of Systems with Security Patterns Using a Fuzzy Approach. In: *Proceedings of the 2006 international conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET*. Springer, 554–564.

Halkidis, S., N. Tsantalis, et al. (July 2008). Architectural Risk Analysis of Software Systems Based on Security Patterns. *IEEE Transactions on Dependable and Secure Computing* 5, 3, 129–142.

Hamid, Brahim, Nicolas Desnos, et al. (2010). Model-based security and dependability patterns in RCES - the TERESA Approach. In: *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems - S&D4RCES '10*. ACM Press.

Hamid, Brahim, Jacob Geisel, et al. (2013). Model-Driven Engineering for Trusted Embedded Systems Based on Security and Dependability Patterns. In: *16th International SDL Forum*. Springer, 72–90.

Hampton, Paul (2012). Survey of safety Architectural Patterns. In: *Achieving Systems Safety*. February 2012. Springer London, 7–9.

Hanmer, Robert S. (2007). *Patterns for Fault Tolerant Software*. Wiley.

Hansen, Kai (Sept. 2009). Security attack analysis of safety systems. *IEEE Conference on Emerging Technologies & Factory Automation*, 1–4.

Harrison, Neil B. and Paris Avgeriou (2008). Incorporating fault tolerance tactics in software architecture patterns. In: *Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems - SERENE '08*. ACM Press.

Howard, M. and D. LeBlanc (2003). *Writing Secure Code*. Microsoft Press.

Johnson, C. and A. Yepez (2011a). Cyber Security Threats to Safety-Critical Space-Based Infrastructures. In: *Proceedings of the Fifth Conference of the International Association for the Advancement of Space Safety*. 1.

Johnson, C. and A. Yepez (2011b). Mapping the Impact of Security Threats on Safety-Critical Global Navigation Satellite Systems. In: *Proceedings of the 29th International Systems Safety Society*. 1. International Systems Safety Society.

Kelly, Tim and Rob Weaver (2004). The Goal Structuring Notation, A Safety Argument Notation. In: *Proceedings of the Dependable Systems and Networks Conference.*

Kumar, Kiran and T.V. Prabhakar (2010a). Design Decision Topology Model for Pattern Relationship Analysis. In: *1st Asian Conference on Pattern Languages of Programs (AsianPLoP 2010).*

Kumar, Kiran and T.V. Prabhakar (2010b). Pattern-oriented Knowledge Model for Architecture Design. In: *17th Conference on Pattern Languages of Programs (PLoP).*

Nai-Fovino, Igor, Marcelo Masera, and Alessio De-Cian (Sept. 2009). Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety* 94, 9, 1394–1402.

Olivera, Andre Rodrigues (2012). Taim : A Safety Pattern Repository, BsC thesis. Federal University of Rio Grande do sul.

Powel, Bruce Douglass (2013). Software Engineering for Embedded Systems. In: Elsevier. Chap. Software Design Architecture Patterns for Embedded Systems.

Preschern, Christopher, Nermin Kajtazovic, and Christian Kreiner (2012a). Built-in security enhancements for the 1oo2 safety architecture. In: *International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER).* IEEE, 103–108.

Preschern, Christopher, Nermin Kajtazovic, and Christian Kreiner (2012b). Catalog of Security Tactics linked to Common Criteria Requirements. In: *19th Conference on Pattern Languages of Programs (PLoP).*

Preschern, Christopher, Nermin Kajtazovic, and Christian Kreiner (2013a). Applying and Evaluating Architectural IEC 61508 Safety Patterns. In: *5th International Conference on Software Technology and Engineering (ICSTE).*

Preschern, Christopher, Nermin Kajtazovic, and Christian Kreiner (2013b). Building a Safety Architecture Pattern System). In: *18th European Conference on Pattern Languages of Programs (EuroPLoP).*

Preschern, Christopher, Nermin Kajtazovic, and Christian Kreiner (2013c). Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle. In: *VikingPLoP.*

Preschern, Christopher, Nermin Kajtazovic, and Christian Kreiner (2013d). Security Analysis of Safety Patterns. In: *20th Conference on Pattern Languages of Programs (PLoP).*

Pullum, Laura (2001). *Software fault tolerance techniques and implementation.* Artech House.

Rauhamäki, Jari and Seppo Kuikka (2013). Patterns for control system safety. In: *18th European Conference on Pattern Languages of Programs (VikingPLoP).*

Rauhamäki, Jari, Timo Vepsäläinen, and Seppo Kuikka (2012). Architectural patterns for functional safety. In: *Nordic Conference on Pattern Languages of Programs (VikingPLoP).*

Rauhamäki, Jari, Timo Vepsäläinen, and Seppo Kuikka (2013). Patterns for safety and control system cooperation. In: *Nordic Conference on Pattern Languages of Programs (VikingPLoP).*

Ryoo, Jungwoo, Phil Laplante, and Rick Kazman (2010). A Methodology for Mining Security Tactics from Security Patterns. In: *2010 43rd Hawaii International Conference on System Sciences.* IEEE, 1–5.

Saridakis, Titos (2002). A System of Patterns for Fault Tolerance. In: *EuroPLoP.*

Sarma, U V R, Sahith Rampelli, and P Premchand (2013). A Catalog of Architectural Design Patterns for Safety-Critical Real-Time Systems. *International Journal of Engineering Research and Applications* 3, 1, 125–131.

Schaad, Andreas and Mike Borozdin (2012). TAM2: Automated Threat Analysis. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing.* ACM, 1103–1108.

Schaad, Andreas and Alexandr Garaga (2012). Automating architectural security analysis. In: *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies.* ACM, 131–132.

Schumacher, Markus (2003). *Security Engineering with Patterns.* Springer.

Spriggs, John (2012). *GSN, The Goal Structuring Notation: A Structured Approach to Presenting Arguments.* Springer.

Steiner, Max and Peter Liggesmeyer (2013). Combination of Safety and Security Analysis - Finding Security Problems That Threaten The Safety of a System. In: *32nd International Conference on Computer Safety, Reliability and Security.* Springer.

Ugljesa, Evzudin and Hans-dieter Wacker (2011). Modeling Security Aspects in Safety Environment. In: *7th International Conference on Electrical and Electronics Engineering*, 46–50.

Wu, Weihang (2003). Safety Tactics for Software Architecture Design. MA thesis. The University of York.

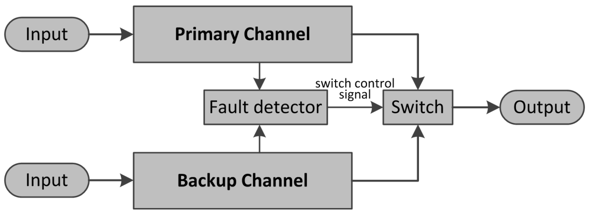
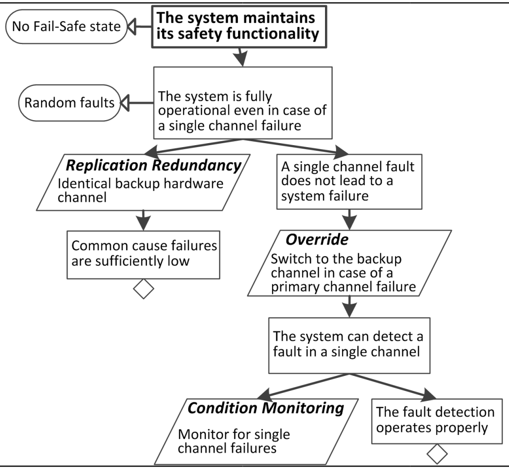
Wu, Weihang (2007). Architectural Reasoning for Safety- Critical Software Applications. PhD thesis. University of York.

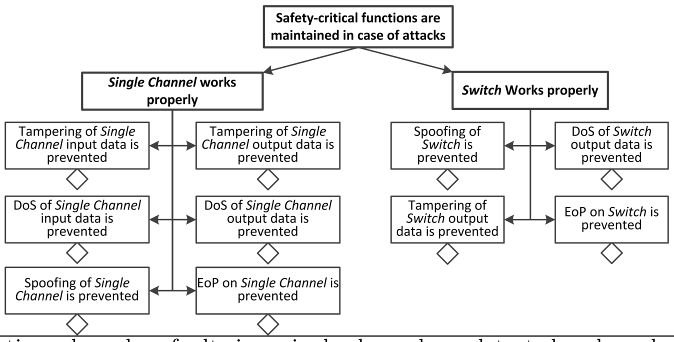
Yampolskiy, Mark et al. (Aug. 2012). Systematic analysis of cyber-attacks on CPS-evaluating applicability of DFD-based approach. In: *5th International Symposium on Resilient Control Systems.* IEEE, 55–62.

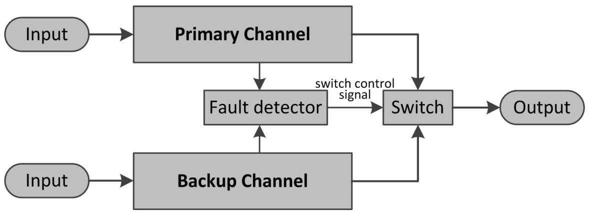
Yautsiukhin, Artsiom and Riccardo Scandariato (2008). Towards a quantitative assessment of security in software architectures. In: *13th Nordic Workshop on Secure IT Systems (NordSec).*

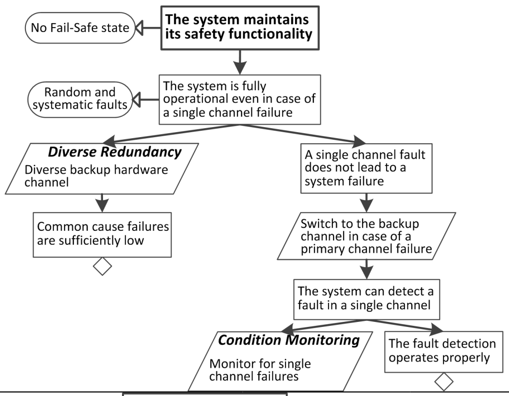
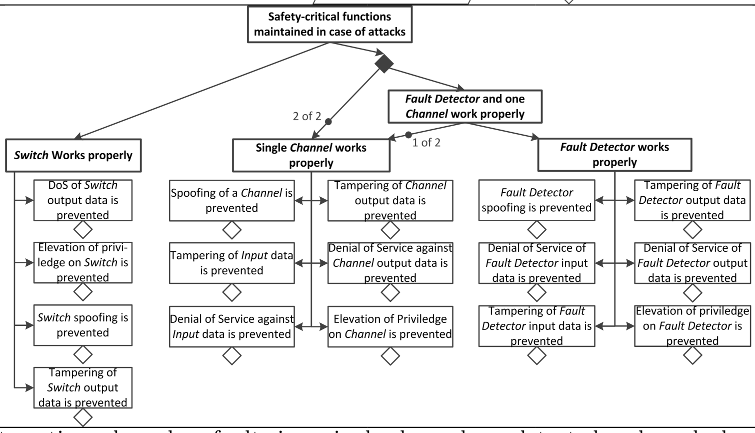
A Security Enhanced Safety Pattern System

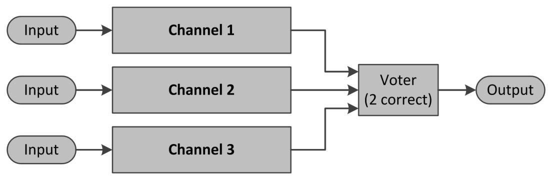
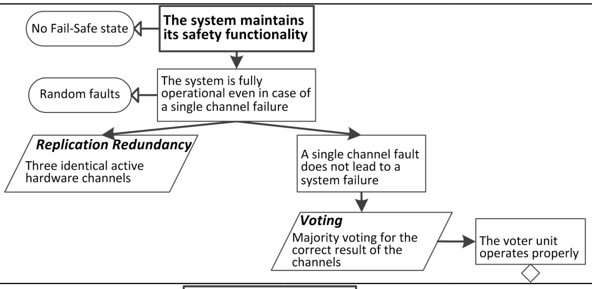
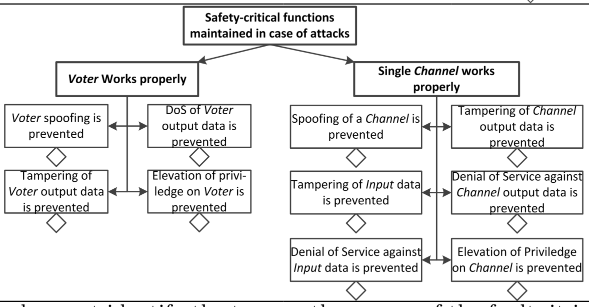
The patterns in our safety pattern system are mostly taken from (Armoush, 2010), because these patterns already provide a comprehensive collection of other patterns in literature and they focus on rather large-scale architectural design decisions which is the main focus of our pattern system. We included all but one of Armoush's patterns. We excluded one pattern (Recovery Block With Backup Voting), because we could not find any known uses for it. Additionally to Armoush's patterns we included the M-out-of-N and the M-out-of-N-D pattern, which are based on architectures described in the IEC 61508 safety standard. For each of the patterns from Figure 6 (page 13), we present the full pattern.

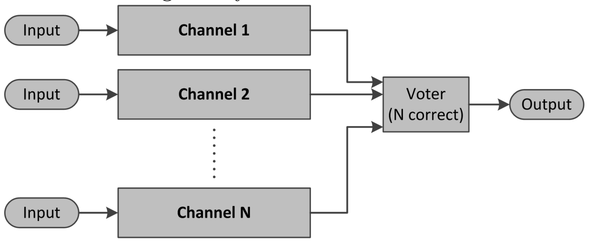
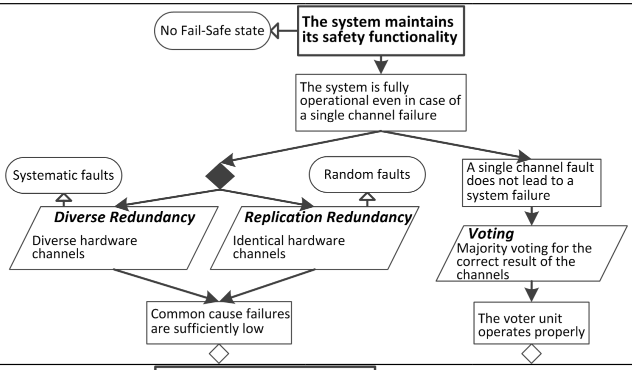
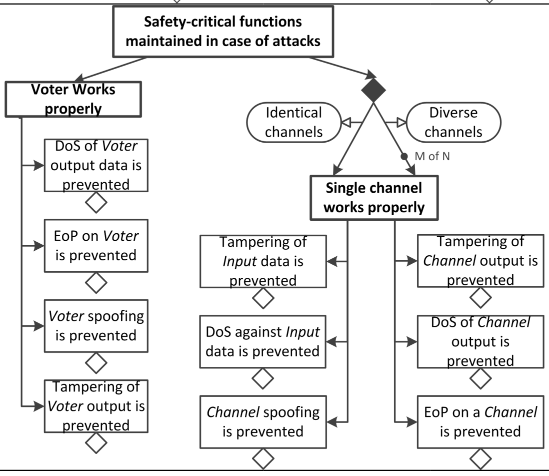
Pattern Name	Homogenous Duplex Pattern **Pattern Type** hardware, failover
AlsoKnownAs	Homogeneous Redundancy Pattern, Standby-Spare Pattern, Dynamic Redundancy Pattern, Two-Channel Redundancy Pattern, 1oo2D Pattern
Context	A safety-critical application without a fail-safe state has a high random error rate and a low systematic error rate.
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components
Forces	- the system cannot shut down because it has no safe state - development costs should not increase - safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime
Solution	The system consists of a *Primary Channel* (active) and a *Secondary Channel* (backup) which are two identical hardware modules. A *Fault detector* monitors the channels and controls a *Switch* to select the *Backup Channel* in case of a *Primary Channel* failure.  Input Primary Channel switch control signal Fault detector Switch Output Input Backup Channel
Safety GSN	 No Fail-Safe state The system maintains its safety functionality Random faults The system is fully operational even in case of a single channel failure Replication Redundancy Identical backup hardware channel A single channel fault does not lead to a system failure Common cause failures are sufficiently low Override Switch to the backup channel in case of a primary channel failure The system can detect a fault in a single channel Condition Monitoring Monitor for single channel failures The fault detection operates properly

Security GSN		 Safety-critical functions are maintained in case of attacks *Single Channel* works properly *Switch* Works properly Tampering of *Single Channel* input data is prevented Tampering of *Single Channel* output data is prevented DoS of *Single Channel* input data is prevented DoS of *Single Channel* output data is prevented Spoofing of *Single Channel* is prevented EoP on *Single Channel* is prevented Spoofing of *Switch* is prevented DoS of *Switch* output data is prevented Tampering of *Switch* output data is prevented EoP on *Switch* is prevented	
Consequences		Systematic and random faults in a single channel are detected and masked. System reliability strongly depends on the fault coverage of the fault detection unit and on the proper functionality of the switch.	
		Affected Attributes	
		Positively	**Negatively**
		Safety: Random Errors in a single channel are handled *Availability:* The full system functionality is still available in case of a single random fault *Maintenance:* Hardware channels can be maintained at runtime	Double hardware costs for system replication
General Scenarios	**SC1**	The system is fully operational even in case of a single channel failure.	
	SC2	A single channel random fault does not lead to a system failure.	
	SC3	The system can detect a fault in a single channel.	
Known Uses		- TOYOPUC-PCS PLC (Miyawaki, 2008) - Navigation system safety (Ljosland, 2006) - Gebhardt GA DUPLEX-S 1oo2D PLC - http://www.gebhardt-automation.com	
Credits		(Douglass, 2002) introduces the pattern. (Grunske, 2003) presents a more general version of this pattern and (Armoush, 2010) adds detailed information about quality attribute related consequences.	

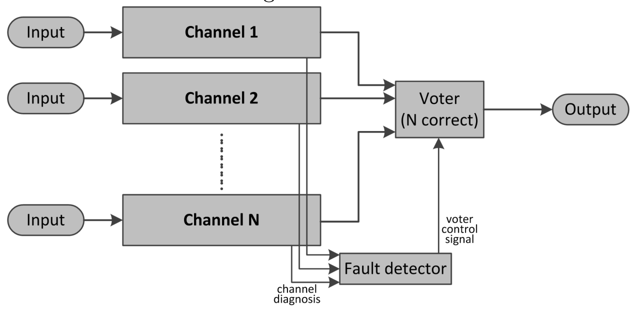
Pattern Name	Heterogenous Duplex Pattern	**Pattern Type**	hardware, failover
AlsoKnownAs	Heterogenous Redundancy Pattern, Diverse Redundancy Pattern, 1oo2D Pattern		
Context	A safety-critical application without a fail-safe state has a high random and systematic error rate.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components		
Forces	- the system cannot shut down because it has no safe state - high safety certification levels require handling of systematic faults - safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime		
Solution	The system consists of a *Primary Channel* (active) and a *Secondary Channel* (backup) which are two diverse hardware modules. A *Fault detector* monitors the channels and controls a *Switch* to select the *Backup Channel* in case of a *Primary Channel* failure.  Input Primary Channel switch control signal Fault detector Switch Output Input Backup Channel		

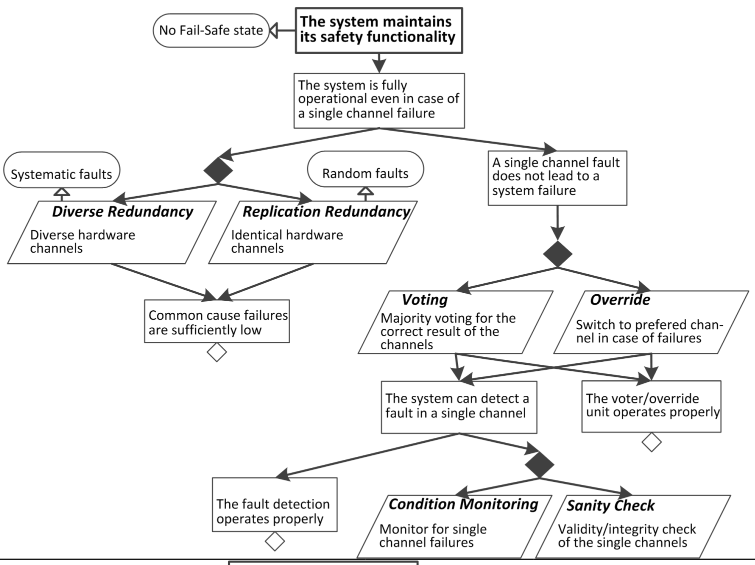
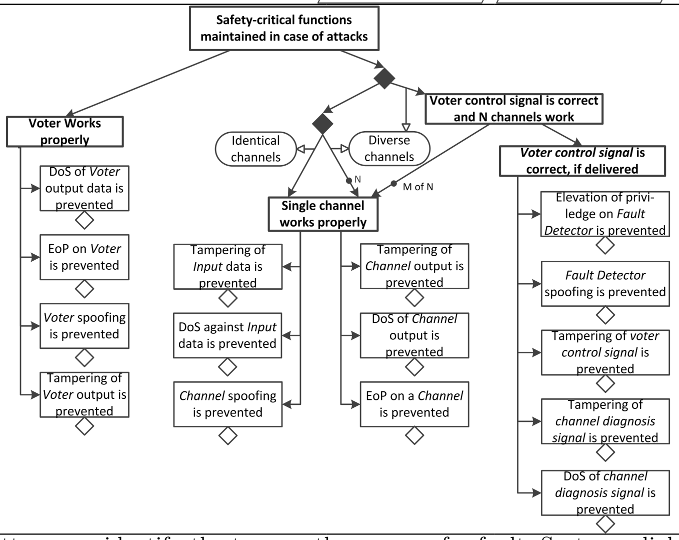
Safety GSN		 No Fail-Safe state The system maintains its safety functionality Random and systematic faults The system is fully operational even in case of a single channel failure Diverse Redundancy Diverse backup hardware channel A single channel fault does not lead to a system failure Common cause failures are sufficiently low Switch to the backup channel in case of a primary channel failure The system can detect a fault in a single channel Condition Monitoring Monitor for single channel failures The fault detection operates properly	
Security GSN		 Safety-critical functions maintained in case of attacks 2 of 2 Fault Detector and one Channel work properly 1 of 2 Switch Works properly Single Channel works properly Fault Detector works properly DoS of Switch output data is prevented Elevation of privi-ledge on Switch is prevented Switch spoofing is prevented Tampering of Switch output data is prevented Spoofing of a Channel is prevented Tampering of Channel output data is prevented Tampering of Input data is prevented Denial of Service against Channel output data is prevented Denial of Service against Input data is prevented Elevation of Priviledge on Channel is prevented Fault Detector spoofing is prevented Tampering of Fault Detector output data is prevented Denial of Service of Fault Detector input data is prevented Denial of Service of Fault Detector output data is prevented Tampering of Fault Detector input data is prevented Elevation of priviledge on Fault Detector is prevented	
Consequences		Systematic and random faults in a single channel are detected and masked. System reliability strongly depends on the fault coverage of the fault detection unit, on the proper functionality of the switch, and on the level of diversity between the two channels	
		Affected Attributes	
		Positively	**Negatively**
		Safety: Random and systematic faults in a single channel are detected and handled *Availability:* The full system functionality is still available in case of a single random or systematic fault *Maintenance:* Hardware channels can be maintained at runtime	Double hardware costs for system replication, Double development costs due to diverse channels, Modifying the functionality of a channel requires double effort
General Scenarios	**SC1**	The system is fully operational even in case of a single channel failure.	
	SC2	A single channel random fault does not lead to a system failure.	
	SC3	A single channel systematic fault does not lead to a system failure.	
	SC4	The system can detect a fault in a single channel.	
Known Uses		- Turbine control system (Kohanawa et al., 2010) - Motor control software (Mutlu, 2004) - YOKOGAWA ProSafe PLCs - http://www.yokogawa.com	
Credits		(Douglass, 2002) introduces the pattern. (Grunske, 2003) presents a more general version of this pattern and (Armoush, 2010) adds detailed information about quality attribute related consequences.	

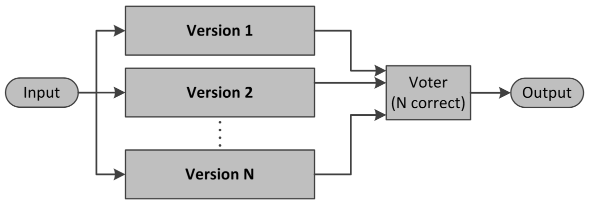
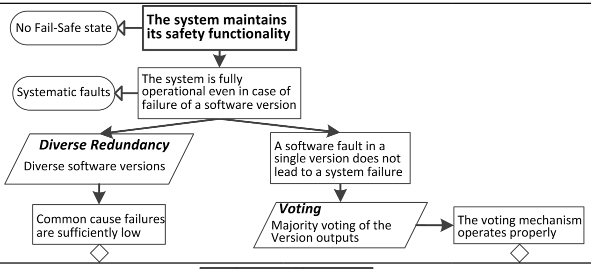
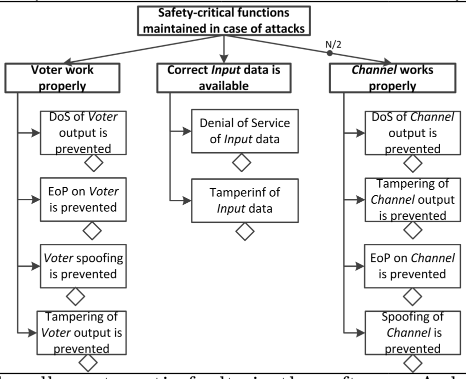
Pattern Name	Triple Modular Redundancy Pattern	**Pattern Type**	hardware, failover
AlsoKnownAs	2oo3 Pattern, Homogeneous Triplex Pattern		
Context	A safety-critical application without a fail-safe state, a high random error and a low systematic error rate.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components.		
Forces	- the system cannot shut down because it has no safe state - safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime		
Solution	Three identical hardware channels operate in parallel. If a single fault occurs in one channel then the other two channels still produce the correct output. A majority voter decides for the correct result.  Input Channel 1 Input Channel 2 Input Channel 3 Voter (2 correct) Output		
Safety GSN	 No Fail-Safe state The system maintains its safety functionality Random faults The system is fully operational even in case of a single channel failure Replication Redundancy Three identical active hardware channels A single channel fault does not lead to a system failure Voting Majority voting for the correct result of the channels The voter unit operates properly		
Security GSN	 Safety-critical functions maintained in case of attacks Voter Works properly Single Channel works properly Voter spoofing is prevented DoS of Voter output data is prevented Tampering of Voter output data is prevented Elevation of privi-ledge on Voter is prevented Spoofing of a Channel is prevented Tampering of Channel output data is prevented Tampering of Input data is prevented Denial of Service against Channel output data is prevented Denial of Service against Input data is prevented Elevation of Priviledge on Channel is prevented		
Consequences	This pattern does not identify the type or the reason of the fault; it just determines the module that contains a fault without correcting the fault itself. The voter has to be very reliable.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Random faults in a single channel are masked *Availability:* The full system functionality is still available in case of a single random fault *Maintenance:* Hardware channels can be maintained at runtime	Triple hardware costs for system replication	
General Scenarios	**SC1**	The system is fully operational even in case of a single channel failure.	
	SC2	A single channel random fault does not lead to a system failure.	
Known Uses	- Turbine control sensor input (Kohanawa et al., 2010) - SRAM applying TMR (Kyriakoulakos and Pnevmatikatos, 2009) - TMS-1000R Gas Turbine - http://www.turbinetech.com		
Credits	(Douglass, 2002) formulates this well-known architecture as a pattern. (Armoush, 2010) adds detailed information about quality attribute related consequences.		

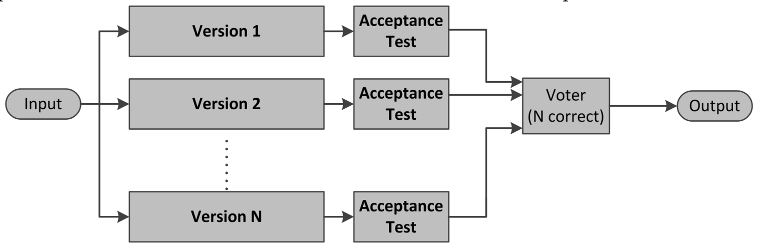
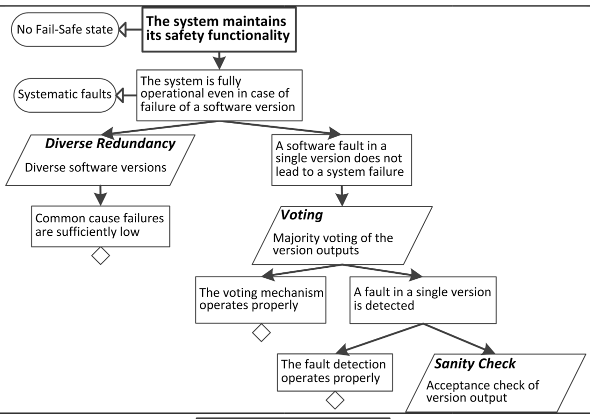
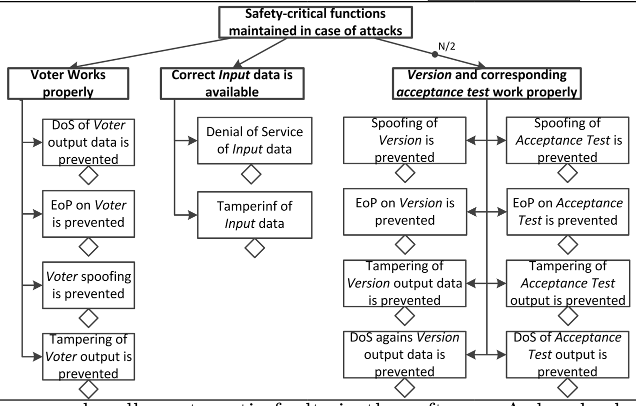
Pattern Name	M-out-of-N Pattern	**Pattern Type**	hard/software, failover
AlsoKnownAs	M/N Parallel Redundancy Pattern, MooN Pattern		
Context	A safety-critical application without a fail-safe state has a high random error rate and a low or high systematic error rate.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components.		
Forces	- the system cannot shut down because it has no safe state - high safety certification levels require handling of systematic faults - safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime		
Solution	N identical or diverse channels (software or hardware) operate in parallel. If a fault occurs in one channel then the other channels still produce the correct output. A voter decides for the result given by at least M channels.  Input Channel 1 Input Channel 2 Input Channel N Voter (N correct) Output		
Safety GSN	 No Fail-Safe state The system maintains its safety functionality The system is fully operational even in case of a single channel failure Systematic faults Random faults A single channel fault does not lead to a system failure Diverse Redundancy Diverse hardware channels Replication Redundancy Identical hardware channels Voting Majority voting for the correct result of the channels Common cause failures are sufficiently low The voter unit operates properly		
Security GSN	 Safety-critical functions maintained in case of attacks Voter Works properly DoS of Voter output data is prevented EoP on Voter is prevented Voter spoofing is prevented Tampering of Voter output is prevented Identical channels Diverse channels M of N Single channel works properly Tampering of Input data is prevented DoS against Input data is prevented Channel spoofing is prevented Tampering of Channel output is prevented DoS of Channel output is prevented EoP on a Channel is prevented		

Consequences		This pattern does not identify the type or the reason of the fault; it just determines the module that contains a fault without correcting the fault itself. To achieve high reliability, the voter has to be very reliable.	
		Affected Attributes	
		Positively	**Negatively**
		Safety: Single-channel random or systematic faults are masked *Availability:* The full system functionality is still available in case of a single fault *Maintenance:* Hardware channels can be maintained at runtime	Multiple hardware costs for system replication and multiple development costs if diverse channels are used
General Scenarios	**SC1**	The system is fully operational even in case of a single channel failure.	
	SC2	A single channel random fault does not lead to a system failure.	
	SC3	A single channel systematic fault does not lead to a system failure.	
Known Uses	- 1oo2 Architecture for LHC detectors (Vergara-Fernandez and Denz, 2002) - Steering system controller (Börcsök et al., 2011) - Netherlocks safety lock - http://halmapr.com/news/netherlocks/tag/3oo4/		
Credits	(Grunske, 2003) describes this pattern and calls it Multi-Channel-Redundancy with Voting. (Armoush, 2010) adds detailed information about quality attribute related consequences.		

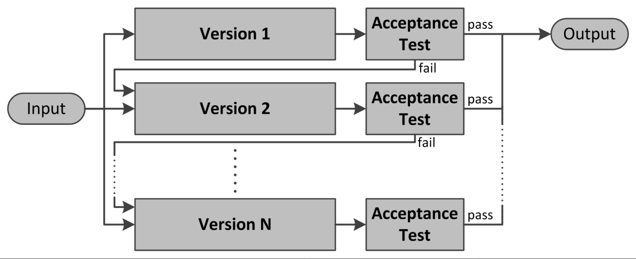
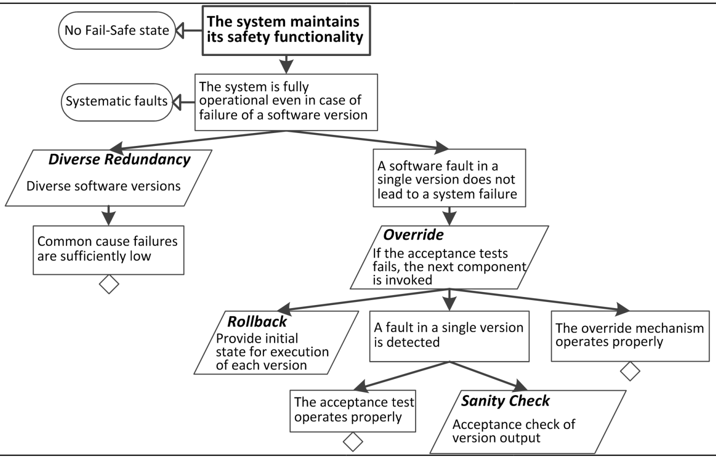
Pattern Name	M-out-of-N-D Pattern	**Pattern Type**	hard/software, failover
AlsoKnownAs	MooN-D Pattern		
Context	A safety-critical application without a fail-safe state has a high random error rate and a low or high systematic error rate.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components.		
Forces	- the system cannot shut down because it has no safe state - high safety certification levels require handling of systematic faults - safety standard requires high fault coverage for single-point of failure components - due to these high availability requirements the hardware platforms must be maintained at the runtime of the system		
Solution	N identical or diverse channels operate in parallel. If a single fault occurs in one channel then the other channels still produce the correct output. A *Voter* decides for the result given by at least M channels. The *Voter* can be influenced by a diagnostic check implemented within the channels. For example, a channel could be excluded from the vote if its diagnostic check fails.  Input Channel 1 Input Channel 2 Voter (N correct) Output Input Channel N voter control signal Fault detector channel diagnosis		

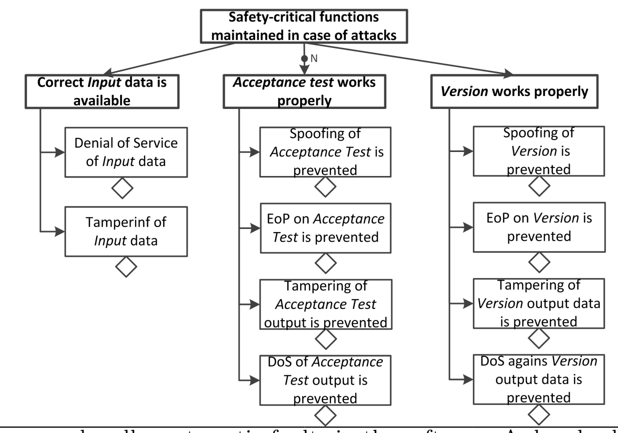
Safety GSN	 No Fail-Safe state The system maintains its safety functionality The system is fully operational even in case of a single channel failure Systematic faults Random faults A single channel fault does not lead to a system failure Diverse Redundancy Diverse hardware channels Replication Redundancy Identical hardware channels Common cause failures are sufficiently low Voting Majority voting for the correct result of the channels Override Switch to prefered chan-nel in case of failures The system can detect a fault in a single channel The voter/override unit operates properly The fault detection operates properly Condition Monitoring Monitor for single channel failures Sanity Check Validity/integrity check of the single channels		
Security GSN	 Safety-critical functions maintained in case of attacks Voter Works properly Voter control signal is correct and N channels work Identical channels Diverse channels N M of N Single channel works properly Voter control signal is correct, if delivered DoS of Voter output data is prevented EoP on Voter is prevented Voter spoofing is prevented Tampering of Voter output is prevented Tampering of Input data is prevented DoS against Input data is prevented Channel spoofing is prevented Tampering of Channel output is prevented DoS of Channel output is prevented EoP on a Channel is prevented Elevation of privi-ledge on Fault Detector is prevented Fault Detector spoofing is prevented Tampering of voter control signal is prevented Tampering of channel diagnosis signal is prevented DoS of channel diagnosis signal is prevented		
Consequences	This pattern can identify the type or the reason of a fault. System reliability strongly depends on the voter unit, the diagnostic test, and the level of diversity between the two channels.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Random or systematic faults in a single channel are masked *Availability:* The full system functionality is still available in case of a single fault *Maintenance:* Hardware channels can be maintained at runtime	Multiple hardware costs for system replication and multiple development costs if diverse channels are used	
General Scenarios	**SC1**	The system is fully operational even in case of a single channel failure.	
	SC2	A single channel random fault does not lead to a system failure.	
	SC3	A single channel systematic fault does not lead to a system failure.	
	SC4	The system can detect a fault in a single channel.	
Known Uses	- HIMA HiQuad 2oo4D architecture (Skambraks, 2006) - 2oo3D architecture for PLC (Alvarez et al., 2005) - Process heater controller - http://rtpcorp.com/documents/ProcessHeaters3000.pdf		
Credits	The MooN-D architecture is described by the IEC 61508 standard.		

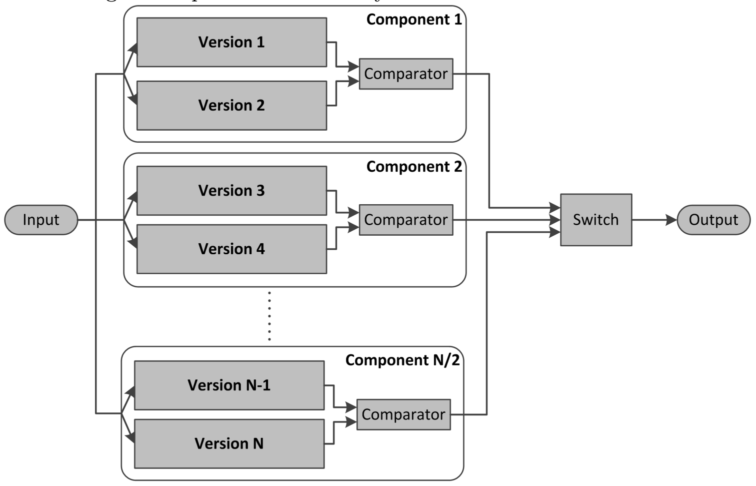
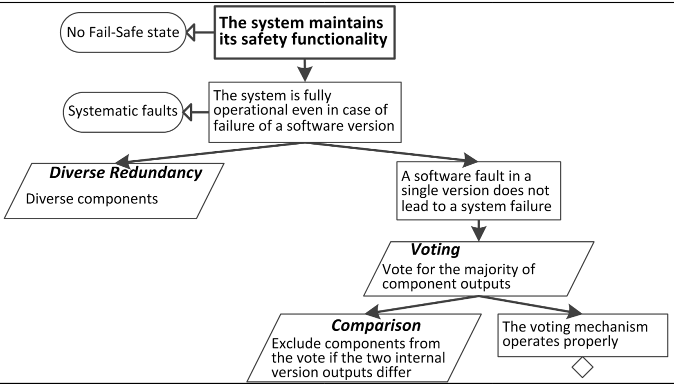
Pattern Name	N-Version Programming Pattern	**Pattern Type**	software, failover
AlsoKnownAs	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - safety standard requires high fault coverage for single-point of failure components		
Solution	N software versions are developed independently from the same specification. The outputs of these versions are sent to the *Voter* which determines the best output.  Input Version 1 Version 2 Version N Voter (N correct) Output		
Safety GSN	 The system maintains its safety functionality No Fail-Safe state The system is fully operational even in case of failure of a software version Systematic faults Diverse Redundancy Diverse software versions A software fault in a single version does not lead to a system failure Common cause failures are sufficiently low Voting Majority voting of the Version outputs The voting mechanism operates properly		
Security GSN	 Safety-critical functions maintained in case of attacks N/2 Voter work properly Correct Input data is available Channel works properly DoS of Voter output is prevented EoP on Voter is prevented Voter spoofing is prevented Tampering of Voter output is prevented Denial of Service of Input data Tamperinf of Input data DoS of Channel output is prevented Tampering of Channel output is prevented EoP on Channel is prevented Spoofing of Channel is prevented		
Consequences	This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions. The voter has to be highly reliable.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Software faults are handled but not detected *Availability:* The full system functionality is still available in case of faults	*Costs:* Multiple development costs, multiple hardware costs if the software versions run on separate hardware *Modifications:* Multiple effort for software modifications	
General Scenarios	**SC1**	The system is fully operational even in case of a failure of a software version.	
	SC2	A software fault in a single version does not lead to a system failure.	
Known Uses	- Analysis of N-version programming (Brilliant et al., 1990) - Train Control (Carr et al., 2005) - Railway Interlocking System (Durmuù et al., 2011)		
Credits	(Armoush, 2010) presents this pattern with detailed information about quality attribute related consequences.		

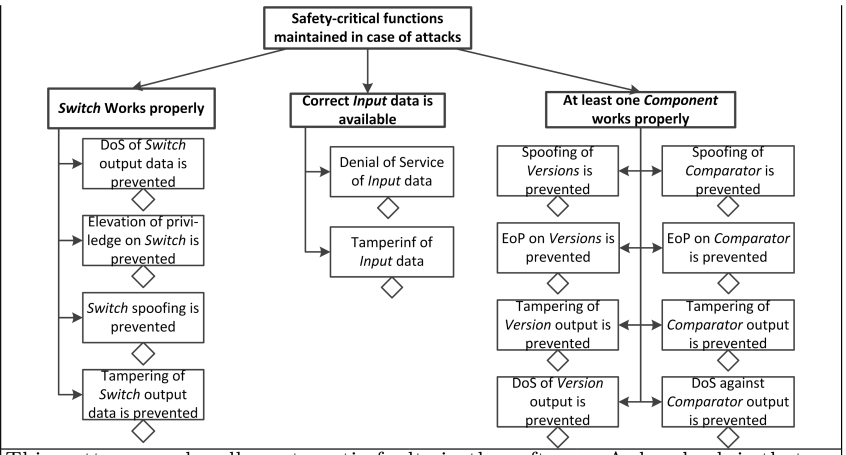
Pattern Name	Acceptance Voting Pattern	**Pattern Type**	software, failover
AlsoKnownAs	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - safety standard requires high fault coverage for single-point of failure components		
Solution	N software versions are developed independently from the same initial specification. The outputs of these versions are checked by an *Acceptance Test* and valid outputs are sent to a *Voter* which determines the best output.  Input Version 1 Version 2 Version N Acceptance Test Acceptance Test Acceptance Test Voter (N correct) Output		
Safety GSN	 No Fail-Safe state The system maintains its safety functionality Systematic faults The system is fully operational even in case of failure of a software version Diverse Redundancy Diverse software versions A software fault in a single version does not lead to a system failure Common cause failures are sufficiently low Voting Majority voting of the version outputs The voting mechanism operates properly A fault in a single version is detected The fault detection operates properly Sanity Check Acceptance check of version output		
Security GSN	 Safety-critical functions maintained in case of attacks N/2 Voter Works properly Correct Input data is available Version and corresponding acceptance test work properly DoS of Voter output data is prevented EoP on Voter is prevented Voter spoofing is prevented Tampering of Voter output is prevented Denial of Service of Input data Tamperinf of Input data Spoofing of Version is prevented EoP on Version is prevented Tampering of Version output data is prevented DoS agains Version output data is prevented Spoofing of Acceptance Test is prevented EoP on Acceptance Test is prevented Tampering of Acceptance Test output is prevented DoS of Acceptance Test output is prevented		
Consequences	This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Software faults are handled and probably detected *Availability:* The full system functionality is still available in case of faults	*Costs:* Multiple development costs, multiple hardware costs if the software versions run on separate hardware *Modifications:* Multiple effort for software modifications	

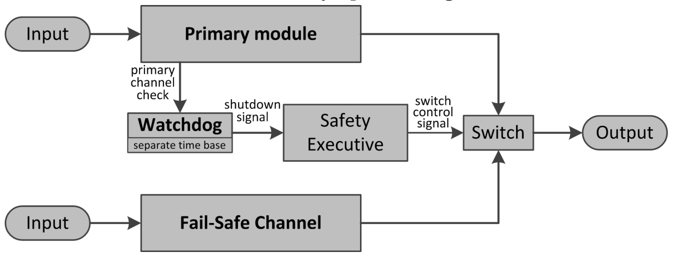
General Scenarios	**SC1**	The system is fully operational even in case of a failure of a software version.
	SC2	A software fault in a single version does not lead to a system failure.
	SC3	A fault in a single software version is detected.
Known Uses	- Dependable web services (Nourani and Azgomi, 2009) - Protected C++ Dispatcher (Borchert et al., 2012) - Fault-tolerant middleware (Kim, 1998)	
Credits	(Armoush, 2010) presents this pattern with detailed information about quality attribute related consequences.	

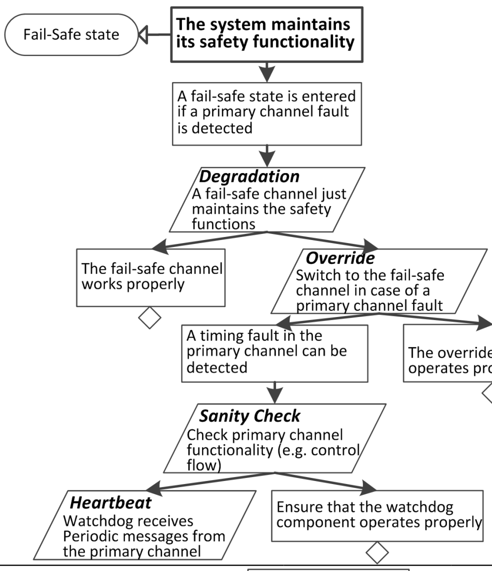
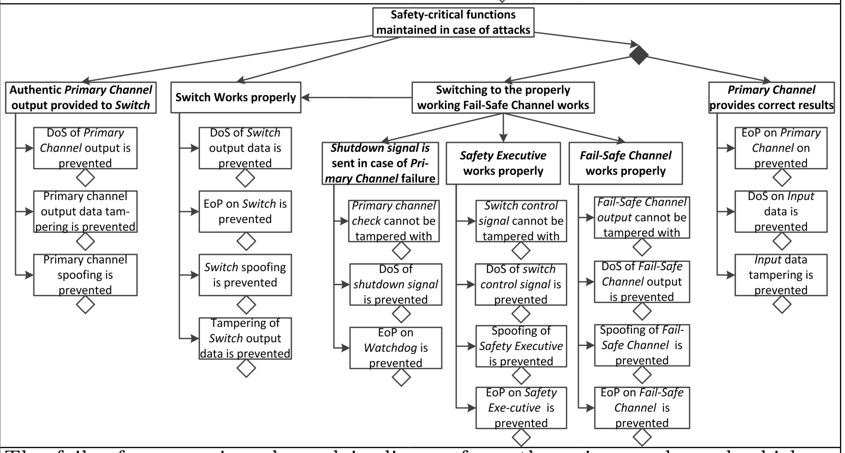
Pattern Name	RECOVERY BLOCK PATTERN	**Pattern Type**	software, failover
AlsoKnownAs	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - safety standard requires high fault coverage for single-point of failure components - no additional processing hardware or processing time is available		
Solution	N software versions are developed independently from the same initial specification. Only a single version is executed at a time. After the execution of *Version 1*, an *Acceptance Test* is executed to check if the software output is reasonable. If the *Acceptance Test* is passed, then the outcome is considered as correct. Otherwise, the system state is restored to its original state and an alternate version is invoked.  Input Version 1 Version 2 Version N Acceptance Test Acceptance Test Acceptance Test pass pass pass fail fail Output		
Safety GSN	 The system maintains its safety functionality No Fail-Safe state The system is fully operational even in case of failure of a software version Systematic faults Diverse Redundancy Diverse software versions Common cause failures are sufficiently low A software fault in a single version does not lead to a system failure Override If the acceptance tests fails, the next component is invoked Rollback Provide initial state for execution of each version A fault in a single version is detected The override mechanism operates properly The acceptance test operates properly Sanity Check Acceptance check of version output		

Security GSN		 Safety-critical functions maintained in case of attacks N Correct *Input* data is available *Acceptance test* works properly *Version* works properly Denial of Service of *Input* data Tamperinf of *Input* data Spoofing of *Acceptance Test* is prevented EoP on *Acceptance Test* is prevented Tampering of *Acceptance Test* output is prevented DoS of *Acceptance Test* output is prevented Spoofing of *Version* is prevented EoP on *Version* is prevented Tampering of *Version* output data is prevented DoS agains *Version* output data is prevented	
Consequences		This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions. Also the reliability highly depends on the quality of the acceptance test.	
		Affected Attributes	
		Positively	**Negatively**
		Safety: Software faults are handled and probably detected *Availability:* The full system functionality is still available in case of faults	*Costs:* Multiple development costs *Modifications:* Multiple effort for software modifications
General Scenarios	**SC1**	The system is fully operational even in case of a failure of a software version.	
	SC2	A software fault in a single version does not lead to a system failure.	
	SC3	A fault in a single software version is detected.	
Known Uses		- Mission-Critical Intrusion-Tolerant Architecture (Wang et al., 2001) - Fault-Tolerant WSN Framework (Beder et al., 2011) - Recovery Block pattern evaluation in software projects (Anderson et al., 1985)	
Credits		(Armoush, 2010) presents this pattern with detailed information about quality attribute related consequences.	

Pattern Name	N-Self Checking Programming Pattern	**Pattern Type**	software, failover
AlsoKnownAs	-		
Context	A safety-critical software without a fail-safe state which probably contains software faults.		
Problem	How to design a system which continues operating even in the presence of software faults.		
Forces	- software often contains faults - high safety certification levels require handling of systematic faults - safety standard requires high fault coverage for single-point of failure components		
Solution	N¿=4 software versions are developed independently from the same initial specification. The versions are arranged in pairs of two as components. Within a component, the results of the two versions are compared to detect errors. If a component fails due to different results from its versions, the next component is invoked to start delivering the required functionality.		
	 Component 1 Version 1 Version 2 Comparator Component 2 Version 3 Version 4 Comparator Input Switch Output Component N/2 Version N-1 Version N Comparator		
Safety GSN	 No Fail-Safe state The system maintains its safety functionality Systematic faults The system is fully operational even in case of failure of a software version Diverse Redundancy Diverse components A software fault in a single version does not lead to a system failure Voting Vote for the majority of component outputs Comparison Exclude components from the vote if the two internal version outputs differ The voting mechanism operates properly		

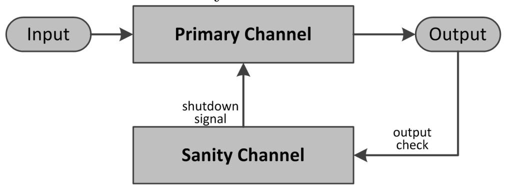
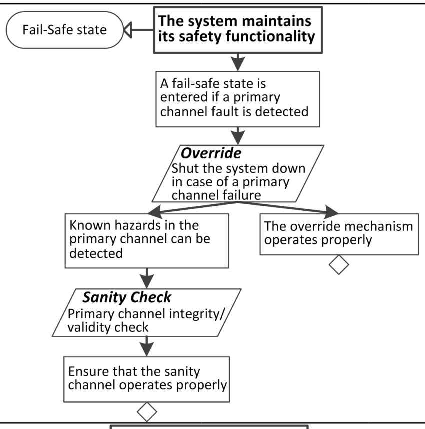
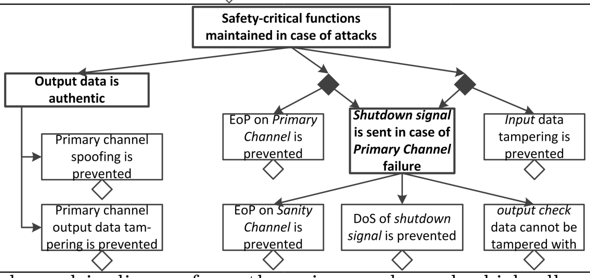
Security GSN		 Safety-critical functions maintained in case of attacks *Switch* Works properly Correct *Input* data is available At least one *Component* works properly DoS of *Switch* output data is prevented Elevation of privi-ledge on *Switch* is prevented *Switch* spoofing is prevented Tampering of *Switch* output data is prevented Denial of Service of *Input* data Tamperinf of *Input* data Spoofing of *Versions* is prevented Spoofing of *Comparator* is prevented EoP on *Versions* is prevented EoP on *Comparator* is prevented Tampering of *Version* output is prevented Tampering of *Comparator* output is prevented DoS of *Version* output is prevented DoS against *Comparator* output is prevented	
Consequences		This pattern can handle systematic faults in the software. A drawback is that the high dependency on the initial specification may lead to a propagation of dependent faults to all versions.	
		Affected Attributes	
		Positively	**Negatively**
		Safety: Software faults are handled and probably detected at component level *Availability:* The full system functionality is still available in case of faults	*Costs:* Multiple development costs *Modifications:* Multiple effort for software modifications
General Scenarios	**SC1**	The system is fully operational even in case of a failure of a software version.	
	SC2	A software fault in a single version does not lead to a system failure.	
Known Uses		- High Available Web services (Parchas and Lemos, 2004) - Airbus uses N-self checking programming (Sghairi et al., 2008) - N-self Checking Programming in the avionics domain (Laprie et al., 1995)	
Credits		(Armoush, 2010) presents this pattern with detailed information about quality attribute related consequences.	

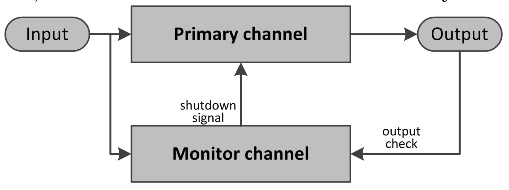
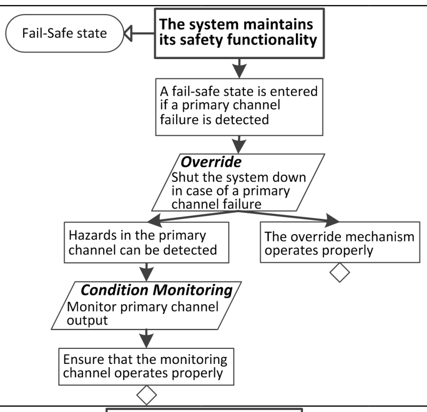
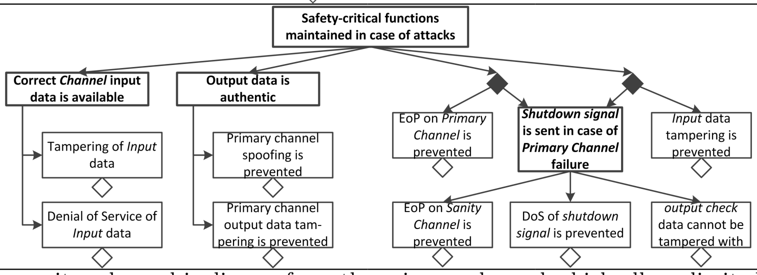
Pattern Name	SAFETY EXECUTIVE PATTERN	**Pattern Type**	hardware, fail-safe
AlsoKnownAs	Safety Kernel Pattern, Shadow-Pattern, Simplex-Pattern		
Context	A system with a complex fail-safe state should maintain its safety functionality even in case of faults.		
Problem	How to check if a fail-safe state should be entered and how to maintain it.		
Forces	- Full redundancy solutions are expensive - An unavailable component cannot tell that it is unavailable - Complex fail-safe state		
Solution	The *Primary Channel* performs all the required functionality. An optional *Fail-Safe Channel* executes just the safety-critical functionality. A centralized *Safety Executive* component coordinates all safety-measures required to shut down the system or to switch over to the *Fail-Safe* processing channel.  Input Primary module primary channel check Watchdog separate time base shutdown signal Safety Executive switch control signal Switch Output Input Fail-Safe Channel		

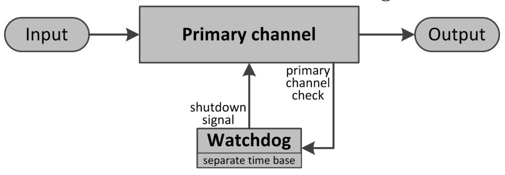
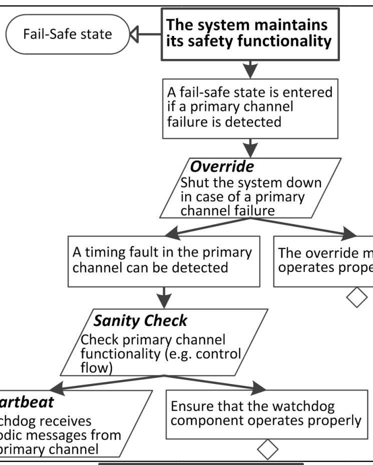
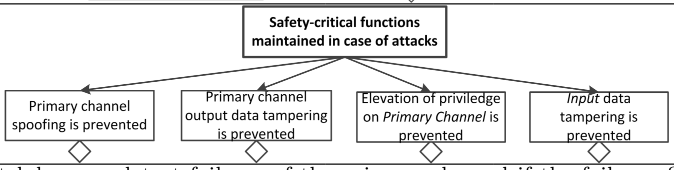
Safety GSN			
Security GSN			
Consequences	The fail-safe processing channel is diverse from the primary channel which allows limited systematic fault detection.		
	Affected Attributes		
	Positively		**Negatively**
	Safety: A safe-state is entered if a fault occurs		*Availability:* Decreased if the system goes into its safe state
General Scenarios	**SC1**	A fail-safe state is entered if a primary channel failure is detected.	
	SC2	A timing fault in the primary channel can be detected.	
Known Uses	- Safety Kernel for Weapon System (Michael et al., 2006) - Cardiac Pacemaker (Bak et al., 2009) - Train Control System (Ghosh et al., 1995)		
Credits	(Douglass, 2002) introduces the pattern and (Armoush, 2010) adds detailed information about quality attribute related consequences.		

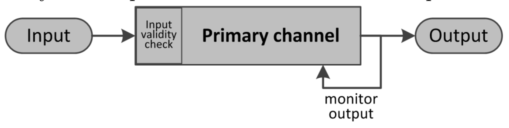
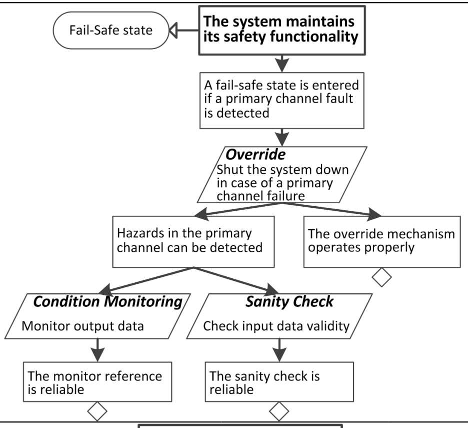
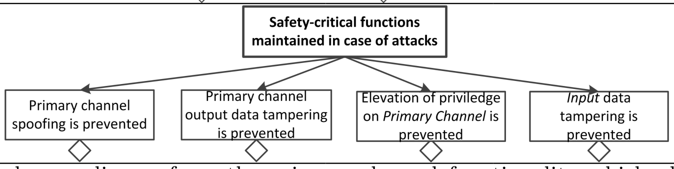
Fail-Safe state
The system maintains its safety functionality
A fail-safe state is entered if a primary channel fault is detected
Degradation
A fail-safe channel just maintains the safety functions
The fail-safe channel works properly
Override
Switch to the fail-safe channel in case of a primary channel fault
A timing fault in the primary channel can be detected
The override mechanism operates properly
Sanity Check
Check primary channel functionality (e.g. control flow)
Heartbeat
Watchdog receives Periodic messages from the primary channel
Ensure that the watchdog component operates properly

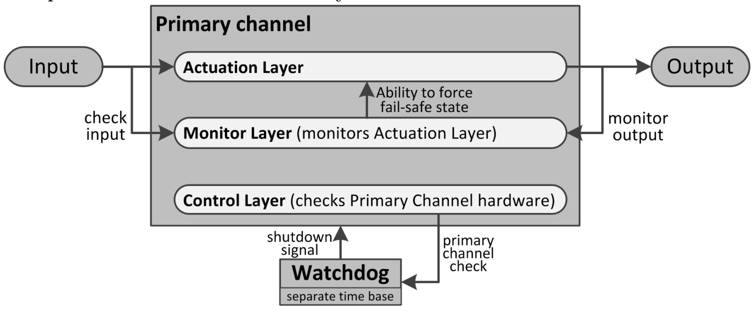
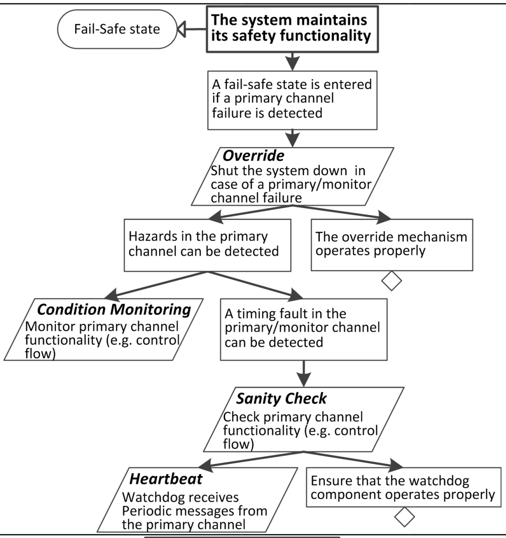
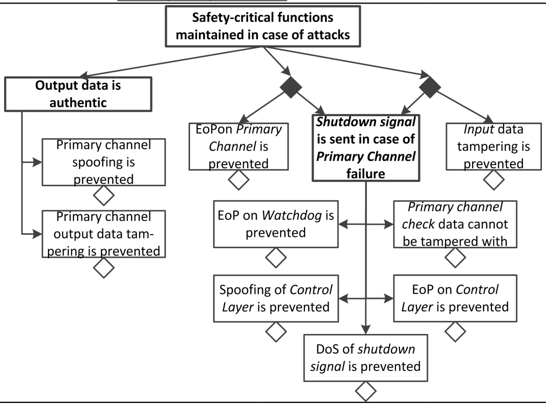
Safety-critical functions maintained in case of attacks
Authentic Primary Channel output provided to Switch
DoS of Primary Channel output is prevented
Primary channel output data tampering is prevented
Primary channel spoofing is prevented
Switch Works properly
DoS of Switch output data is prevented
EoP on Switch is prevented
Switch spoofing is prevented
Tampering of Switch output data is prevented
Switching to the properly working Fail-Safe Channel works
Shutdown signal is sent in case of Primary Channel failure
Primary channel check cannot be tampered with
DoS of shutdown signal is prevented
EoP on Watchdog is prevented
Safety Executive works properly
Switch control signal cannot be tampered with
DoS of switch control signal is prevented
Spoofing of Safety Executive is prevented
EoP on Safety Exe-cutive is prevented
Fail-Safe Channel works properly
Fail-Safe Channel output cannot be tampered with
DoS of Fail-Safe Channel output is prevented
Spoofing of Fail-Safe Channel is prevented
EoP on Fail-Safe Channel is prevented
Primary Channel provides correct results
EoP on Primary Channel on prevented
DoS on Input data is prevented
Input data tampering is prevented

Pattern Name	Sanity Check Pattern	**Pattern Type**	hardware, fail-safe
AlsoKnownAs	-		
Context	A safety-critical system with a fail-safe state and low availability requirements.		
Problem	Find an appropriate mechanism to detect failures or errors that can lead to known hazards.		
Forces	- The set of relevant hazards is often known for a specific application domain - Full redundancy solutions are expensive		
Solution	A separate *Sanity Channel* monitors the correct operation of the *Primary Channel*. If the *Primary Channel* output deviates to much from the expected result, then the *Sanity Channel* shuts the system down.  Input Primary Channel Output shutdown signal output check Sanity Channel		
Safety GSN	 Fail-Safe state The system maintains its safety functionality A fail-safe state is entered if a primary channel fault is detected Override Shut the system down in case of a primary channel failure Known hazards in the primary channel can be detected The override mechanism operates properly Sanity Check Primary channel integrity/ validity check Ensure that the sanity channel operates properly		
Security GSN	 Safety-critical functions maintained in case of attacks Output data is authentic Primary channel spoofing is prevented Primary channel output data tampering is prevented EoP on Primary Channel is prevented Shutdown signal is sent in case of Primary Channel failure Input data tampering is prevented EoP on Sanity Channel is prevented DoS of shutdown signal is prevented output check data cannot be tampered with		
Consequences	The sanity channel is diverse from the primary channel which allows limited systematic fault as well as random fault detection.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Known hazards can be handled	*Availability:* Decreased if the system goes into its safe state	
General Scenarios	**SC1**	A fail-safe state is entered if a primary channel fault is detected.	
	SC2	Known hazards in the primary channel can be detected.	
Known Uses	- Oxygen level software Sanity Channel - PISCAS (Preschern, 2011) - Automotive Distance Sensor (Zimmer, 2009) - Sanity Check in Semiconductor Devices (Tong, 2007)		
Credits	(Douglass, 2002) introduces the pattern. (Grunske, 2003) presents a more general version of this pattern and (Armoush, 2010) adds detailed information about quality attribute related consequences.		

Pattern Name	Monitor-Actuator Pattern	**Pattern Type**	hardware, fail-safe
AlsoKnownAs	-		
Context	A safety-critical system with a fail-safe state and with low availability requirements.		
Problem	Find an appropriate mechanism to detect failures or errors		
Forces	- Full redundancy solutions are expensive		
Solution	A separate *Monitor Channel* monitors the correct operation of the primary channel. The *Monitor Channel* computes reference values from the inputs and compares them to the *Primary Channel* output. If the value deviates to much from the expected result, then the *Monitor Channel* shuts the system down.  Input; Primary channel; Output; shutdown signal; output check; Monitor channel		
Safety GSN	 Fail-Safe state; The system maintains its safety functionality; A fail-safe state is entered if a primary channel failure is detected; *Override* Shut the system down in case of a primary channel failure; Hazards in the primary channel can be detected; The override mechanism operates properly; *Condition Monitoring* Monitor primary channel output; Ensure that the monitoring channel operates properly		
Security GSN	 Safety-critical functions maintained in case of attacks; Correct *Channel* input data is available; Output data is authentic; Tampering of *Input* data; Denial of Service of *Input* data; Primary channel spoofing is prevented; Primary channel output data tampering is prevented; EoP on *Primary Channel* is prevented; *Shutdown signal* is sent in case of *Primary Channel* failure; *Input* data tampering is prevented; EoP on *Sanity Channel* is prevented; DoS of *shutdown signal* is prevented; *output check* data cannot be tampered with		
Consequences	The monitor channel is diverse from the primary channel which allows limited systematic fault as well as random fault detection.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Hazardous situations can be detected and handled	*Availability:* Decreased if the system goes into its safe state	
General Scenarios	**SC1**	A fail-safe state is entered if a primary channel failure is detected.	
	SC2	Hazards in the primary channel can be detected.	
Known Uses	- Fly-by-Wire System (Jacazio et al., 2008) - Vehicle Simulator (Chao et al., 2004) - WSN Testbed (Bapat et al., 2007)		
Credits	(Douglass, 2002) introduces the pattern. (Grunske, 2003) presents a more general version of this pattern and (Armoush, 2010) adds detailed information about quality attribute related consequences.		

Pattern Name	WATCHDOG PATTERN	**Pattern Type**	hardware, fail-safe
AlsoKnownAs	Watchdog Timer, Watchdog Processor, Hardware Watchdog Pattern		
Context	A system provides a timing-critical safety functionality.		
Problem	How to make sure that the internal computational processing is proceeding properly and timely.		
Forces	- Full redundancy solutions are expensive - An unavailable component cannot tell that it is unavailable		
Solution	A separate *Watchdog* hardware component receives liveness messages from the *Primary Channel*. If the *Watchdog* does not receive the expected messages, it will initiate a corrective action such as a shutdown signal.  Input Primary channel Output primary channel check shutdown signal Watchdog separate time base		
Safety GSN	 Fail-Safe state The system maintains its safety functionality A fail-safe state is entered if a primary channel failure is detected Override Shut the system down in case of a primary channel failure A timing fault in the primary channel can be detected The override mechanism operates properly Sanity Check Check primary channel functionality (e.g. control flow) Heartbeat Watchdog receives Periodic messages from the primary channel Ensure that the watchdog component operates properly		
Security GSN	 Safety-critical functions maintained in case of attacks Primary channel spoofing is prevented Primary channel output data tampering is prevented Elevation of priviledge on *Primary Channel* is prevented *Input* data tampering is prevented		
Consequences	The watchdog can detect failures of the primary channel if the failure affects the liveness messages. The watchdog is diverse from the primary channel which allows limited systematic fault detection.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Timing faults are handled	*Availability:* Decreased if the system shuts down	
General	**SC1**	A fail-safe state is entered if a primary channel failure is detected.	
Scenarios	**SC2**	A timing fault in the primary channel can be detected.	
Known Uses	- Medical Robot (Guiochet and Vilchis, 2002) - Traffic Management System (Stögerer and Kastner, 2010) - Software implemented watchdog pattern (Chen et al., 2007)		
Credits	(Douglass, 2002) introduces the pattern. (Grunske, 2003) and (Hanmer, 2007) also present this pattern and (Armoush, 2010) adds detailed information about quality attribute related consequences.		

Pattern Name	PROTECTED SINGLE CHANNEL	**Pattern Type**	hard/software, fail-safe
AlsoKnownAs	Safety Kernel Pattern, Shadow-Pattern, Simplex-Pattern		
Context	A system with a fail-safe state and with low availability requirements.		
Problem	Find an appropriate mechanism to handle failures or errors that can lead to known hazards.		
Forces	- Full redundancy solutions are expensive - Components are so complex that we cannot assume them to be error free - Not any additional hardware components can be introduced		
Solution	The input and/or output data of the *Primary Channel* is monitored and checked regarding its validity or compared to reference data or expected data.  Input Input validity check Primary channel Output monitor output		
Safety GSN	 Fail-Safe state The system maintains its safety functionality A fail-safe state is entered if a primary channel fault is detected Override Shut the system down in case of a primary channel failure Hazards in the primary channel can be detected The override mechanism operates properly Condition Monitoring Monitor output data Sanity Check Check input data validity The monitor reference is reliable The sanity check is reliable		
Security GSN	 Safety-critical functions maintained in case of attacks Primary channel spoofing is prevented Primary channel output data tampering is prevented Elevation of priviledge on Primary Channel is prevented Input data tampering is prevented		
Consequences	The checks are diverse from the primary channel functionality which allows limited systematic fault detection.		
	Affected Attributes		
	Positively	**Negatively**	
	Safety: Known hazards can be handled	*Availability:* Decreased if the system goes into its safe state	
General Scenarios	**SC1**	A fail-safe state is entered if a primary channel fault is detected.	
	SC2	Hazards in the primary channel can be detected.	
Known Uses	- Robot hand (Kumar et al., 2011) - Protected Single Channel implementation in C (Douglass, 2010) - Real-time autonomic system architecture (Solomon et al., 2007)		
Credits	(Douglass, 2002) introduces the pattern. (Grunske, 2003) also presents this pattern and (Armoush, 2010) adds detailed information about quality attribute related consequences.		

Pattern Name	3-LEVEL SAFETY MONITORING	**Pattern Type**	hard/software, fail-safe
AlsoKnownAs	Safety Kernel Pattern, Shadow-Pattern, Simplex-Pattern		
Context	A system with a fail-safe state and with low availability requirements		
Problem	Find an appropriate mechanism to handle failures or errors that can lead to known hazards.		
Forces	- Full redundancy solutions are expensive - Components are so complex that we cannot assume them to be error free		
Solution	Divide the system into 3 layers: - The *Actuation Layer* performs the system functionality - The *Monitoring Layer* monitors the *Actuation Layer* and forces a fail-safe state if values deviate too much from references - The *Control Layer* checks the system hardware and sends messages to a *Watchdog* component which can shut the system down  Primary channel Input Actuation Layer Output Ability to force fail-safe state check input Monitor Layer (monitors Actuation Layer) monitor output Control Layer (checks Primary Channel hardware) shutdown signal primary channel check Watchdog separate time base		
Safety GSN	 Fail-Safe state The system maintains its safety functionality A fail-safe state is entered if a primary channel failure is detected Override Shut the system down in case of a primary/monitor channel failure Hazards in the primary channel can be detected The override mechanism operates properly Condition Monitoring Monitor primary channel functionality (e.g. control flow) A timing fault in the primary/monitor channel can be detected Sanity Check Check primary channel functionality (e.g. control flow) Heartbeat Watchdog receives Periodic messages from the primary channel Ensure that the watchdog component operates properly		
Security GSN	 Safety-critical functions maintained in case of attacks Output data is authentic Primary channel spoofing is prevented Primary channel output data tampering is prevented EoP on Primary Channel is prevented Shutdown signal is sent in case of Primary Channel failure Input data tampering is prevented EoP on Watchdog is prevented Primary channel check data cannot be tampered with Spoofing of Control Layer is prevented EoP on Control Layer is prevented DoS of shutdown signal is prevented		

Consequences		The pattern is not applicable for systems with high availability requirements. The checks are diverse from the primary channel which allows limited systematic fault detection.	
		Affected Attributes	
		Positively	**Negatively**
		Safety: Known hazards can be handled	*Availability:* Decreased if the system goes into its safe state
General Scenarios	**SC1**	A fail-safe state is entered if a primary channel failure is detected.	
	SC2	Hazards in the primary channel can be detected.	
	SC3	A timing fault in the primary/monitor channel can be detected.	
Known Uses	- E-Gas unit to control motor vehicle drive power (Bederna and Zeller, 1999) - Standardized E-Gas concept (EGAS, Arbeitskreis, 2006) - Yokogawa ProSafe-RS PLC (Emori and Kawakami, 2005)		
Credits	(Armoush, 2010) presents this pattern with detailed information about quality attribute related consequences.		

B Safety Tactics

This section presents the full list of safety tactics from (Preschern et al., 2013) (where a more detailed explanation about safety tactics can be found).

Tactic	Aim	Description	IEC 61508 methods
Simplicity	Avoid failures through keeping the system as simple as possible.	*Simplicity* reduces the system complexity. It includes structuring methods or cutting unnecessary functionality and organizes system elements or reduces them to their core safety functionality, thus, eliminating hazards. An example for the application of the *Simplicity* tactic is an emergency stop switch system which is usually kept as simple as possible.	IEC 61508-7: B.2.1 structured specification, B.3.2 structured design, C.2.7 structured programming, E.3 structured description method, C.4.2 programming language subset, C.4.2 limit asynchronous constructs, E.5.13 software complexity controller
Substitution	Avoid failures though usage of more reliable components.	Components or methods are replaced by other components or methods one has higher confidence in. For hardware and software this can mean usage of existing components which are well-proven in the safety domain.	IEC 61508-7: B.3.3 usage of well-proven components, B.5.4 field experience, C.2.10 usage of well-proven/verified software elements, E.20 application of validated soft-cores, E.35 application of validated hard-cores, E.41 usage of well-tried circuits, C.4.3 certified tools and compilers, C.4.4 well-proven tools and compilers, E.4 well-proven tools, E.42 well-proven production process, E.28 application of well-proven synthesis tools, E.29 application of well-proven libraries
Sanity Check (Checking)	Detection of implausible system outputs or states.	The *Sanity Check* tactic checks whether a system state or value remains within a valid range which can be defined in the system specification or which is based on knowledge about the internal structure or nature of the system. An example for a *Sanity Check* is a stuck-at fault RAM-test which checks the proper functionality of the memory during system runtime. The test is based on the understanding of the memory behavior (if we write data to the memory, we should later on be able to read the same data). Faults are detected if the memory behaves differently.	A.1.2 monitoring relay contacts, A.2.7 analog signal monitoring, A.3.1-A.3.3 self-tests, A.4.1-A.4.4 checksums, A.5.1-A.5.5 RAM-Tests, A.6.1 test pattern, A.7.1 one-bit hardware redundancy, A.7.2 multi-bit hardware redundancy, A.7.4 inspection using test patterns, A.9 temporal and logical program monitoring, C.3.3 assertion programming, C.5.3 interface checking, C.4.1 strong typed programming language

Tactic	**Aim**	**Description**	**IEC 61508 methods**
Condition Monitoring (Checking)	Detect deviations from the intended system outputs or states.	*Condition Monitoring* checks whether a system value remains within a reasonable range compared to a more reliable, but usually less accurate, reference value. The reference value is computed at runtime by a redundant part in the implementation which can be based on system input values and is not pre-known from the specification (like it would be the case for *Sanity Check*). An example for *Condition Monitoring* is a system which has to be time-synchronized via the Internet and which checks if the synchronized time is feasible by comparing it to an internal clock.	IEC 61508-7: A.1.1 failure detection by online monitoring, A.6.4 monitored outputs, A.8.2 voltage control, A.9 temporal and logical program monitoring, A.12.1 reference sensor, A.13.1 monitoring
Comparison	Detection of discrepancies of redundant system outputs.	*Comparison* tests if the outputs of fully redundant subsystems are equal in order to detect failures. The *Comparison* tactic usually implies the usage of a redundancy tactic. An example for the application of the *Comparison* tactic is a dual-core processor running in lock-step mode. The processor runs the same software on both cores and compares their outputs after each cycle.	IEC 61508-7: A.1.3 comparator, A.6.5 input comparison/voting
Diverse Redundancy (Redundancy)	Introduction of a redundant system which allows detection or masking of failures in the specification or implementation as well as random hardware failures.	*Diverse Redundancy* can be applied to the specification or to the implementation level. In a system using *Diverse Redundancy* on the implementation level, redundant components use different implementations which were developed independently from the same specification. *Diverse Redundancy* on a specification level goes one step further and additionally requires that even the requirement specifications for the redundant components have to be set up by individual teams.	IEC 61508-7: A.7.6 information redundancy, A.13.2 cross-monitoring of multiple actuators, B.1.4 diverse hardware, C.4.4 diverse programming
Replication Redundancy (Redundancy)	Introduction of a redundant systems which allows detection or masking of random hardware failures (not systematic failures).	*Replication Redundancy* means introduction of a redundant system of the same implementation. The redundant systems maintain the same functionality, use identical hardware, and run the same software implementation. An example for *Replication Redundancy* is the RAID1 data storage technology.	IEC 61508-7: A.2.1 tests by redundant hardware, A.2.5 monitored redundancy, A.3.5 reciprocal comparison by software, A.4.5 block replication, A.6.3 multi-channel output, A.7.3 complete hardware redundancy, A.7.5 transmission redundancy
Repair (Recovery)	Bring a failed system back to a state of full functionality.	The full system functionality is manually or automatically restored if a system failure occurs.	IEC 61508-7: C.3.9 error correction, C.3.10 dynamic reconfiguration

Tactic	Aim	Description	IEC 61508 methods
Degradation (Recovery)	*Degradation* brings a system with an error into a state with reduced functionality in which the system still maintains the core safety functions.	*Degradation* systems define a core safety functionality. The systems maintain this safety functionality and additional non-critical functions. In case of an error, the system falls back into a degraded mode in which it just maintains the core safety functionality. An example where the *Degradation* tactic is often applied are automation systems. These systems control safety-critical processes and often visualize these processes in a GUI. If the system has too few resources (e.g. processing time), then the system stops the GUI service and just focuses on its core functionality to control the safety-critical processes.	IEC 61508-7: A.8 voltage supply error handling, C.3.8 degraded function limitation
Voting (Masking)	Mask the failure of a subsystem so that the failure does not propagate to other systems.	*Voting* makes a failure transparent. The tactic does not try to repair the failure, but it hides the failure through choosing a correct result from redundant subsystems. It decides for the majority of the output values.	IEC 61508-7: A.1.4 voter, A.6.5 input comparison/voting
Override (Masking)	Mask the failure of a subsystem so that the failure does not propagate to other systems.	The *Override* tactic forces the system output to a safe state. For example, if we have a system which is in a safe state when shut off, we can apply the *Override* tactic to shut off the system if we have doubt about the system output (e.g. if an output validity check fails). In this scenario overriding the system output with a safe output value decreases the availability of the system. Another form of the *Override* tactic, which does not decrease the availability and is closely related to the *Voting* tactic, chooses the output of redundant subsystems by preferring one subsystem or one output state over another.	IEC 61508: Fail-Safe Principle, A.1.3 comparator
Barrier	Protect a subsystem from influences or influencing other subsystems.	The *Barrier* tactic provides a mechanism to protect from unintentional influences between subsystems. To apply *Barrier*, the interfaces between subsystems have to be analyzed and specified. These interfaces are controlled at runtime by a trustworthy component (the *Barrier*) which often is an already existing reliable mechanism. An example for a *Barrier* is a memory protection unit which controls and restricts the communication between different tasks.	IEC 61508-7: A.11 separation of energy lines from information lines, B.1.3 separation of safety functions from non-safety functions, B.3.4 modularization, C.2.8 information hiding/ encapsulation, C.2.9 modular approach, E.12 modularization, C.3.11 time-triggered architecture

Appendix References

Alvarez, Jacobo et al. (2005). Safe PLD-based programmable controllers. In: *International Conference on Field Programmable Logic and Applications*. IEEE, 559–562.

Anderson, T. et al. (1985). Software Fault Tolerance: An Evaluation. *IEEE Transactions on Software Engineering* SE-11, 12, 1502–1510.

Armoush, Ashraf (2010). Design patterns for safety-critical embedded systems. PhD thesis. RWTH Aachen University.

Bak, Stanley et al. (Apr. 2009). The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety. In: *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 99–107.

Bapat, Sandip et al. (2007). Chowkidar : A Health Monitor for Wireless Sensor Network Testbeds. In: *3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities (TridentCom)*. IEEE.

Beder, Delano M., Jo Ueyama, and Marcos L. Chaim (Dec. 2011). A generic policy-free framework for fault-tolerant systems: Experiments on WSNs. In: *2011 IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications*. IEEE, 1–7.

Bederna, F. and T. Zeller (1999). *Method and arrangement for controlling the drive unit of a vehicle*.

Borchert, Christoph, Horst Schirmeier, and Olaf Spinczyk (2012). Protecting the Dynamic Dispatch in C ++ by Dependability Aspects. In: *1st GI Workshop on Software-Based Methods for Robust Embedded Systems (SOBRES '12)*.

Börcsök, Josef et al. (2011). High-Availability Controller Concept for Steering Systems: The Degradable Safety Controller. In: *Proceedings of the 2nd international conference on Circuits, Systems, Communications & Computers*, 220–228. isbn: 9781618040565.

Brilliant, S.S., J.C. Knight, and N.G. Leveson (1990). Analysis of faults in an N-version software experiment. *IEEE Transactions on Software Engineering* 16, 2, 238–247.

Carr, D.W. et al. (2005). An Open On-Board CBTC Controller Based on N-Version Programming. In: *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*. Vol. 1. IEEE, 834–839.

Chao, H.C., T.W. Pearce, and M.J.D. Hayes (2004). Use of the HLA in a Real-Time Multi-Vehicle Simulator. In: *The Canadian Society of Mechanical Engineering Forum*, 1–10.

Chen, Xi et al. (2007). Application of Software Watchdog as a Dependability Software Service for Automotive Safety Relevant Systems. In: *37th International Conference on Dependable Systems and Networks (DSN)*. IEEE.

Douglass, Bruce Powel (2002). *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Pearson.

Douglass, Bruce Powel (2010). *Design Patterns for Embedded Systems in C*. Elsevier.

Durmuù, Mustafa Seçkin et al. (2011). A New Voting Strategy in Diverse Programming for Railway Interlocking Systems. In: *International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE)*. IEEE, 723–726.

EGAS, Arbeitskreis (2006). *Standardisiertes E-Gas-Ueberwachungskonzept fuer Motorsteuerungen von Otto- und Dieselmotoren*.

Emori, Toshiyuki and Shigehito Kawakami (2005). Safety technologies incorporated in the safety control system. *Yokogawa Technical Report* 40, 4, 43–46.

Ghosh, A.K. et al. (1995). A distributed safety-critical system for real-time train control. In: *21st Annual Conference on IEEE Industrial Electronics*. Vol. 2. IEEE, 760–767.

Grunske, Lars (2003). Transformational Patterns for the Improvement of Safety Properties in Architectural Specification. In: *Proceedings of The Second Nordic Conference on Pattern Languages of Programs (VikingPLoP)*.

Guiochet, J. and A. Vilchis (2002). Safety Analysis of a Medical Robot for Tele-echography. In: *2nd IARP IEEE/RAS joint workshop on Technical Challenge for Dependable Robots in Human Environments*. IEEE, 217–227.

Hanmer, Robert S. (2007). *Patterns for Fault Tolerant Software*. Wiley.

Jacazio, G., P. Serena Guinzio, and M. Sorli (2008). A dual-duplex electrohydraulic system for the fly-by-wire control of a helicopter main rotor. In: *26th International Congress of the Aeronautical Sciences*, 1–9.

Kim, K H Kane (1998). ROAFTS : A Middleware Architecture for Real-time Object-oriented Adaptive Fault Tolerance Support. In: *3rd International High-Assurance Systems Engineering Symposium*. IEEE.

Kohanawa, Akihiko, Masami Hasegawa, and Shigeharu Kanamori (2010). Safety Control Solutions Protecting Onsite Safety. *Fuji Electric Group* 56, 1.

Kumar, S Phani, P. Seetha Ramaiah, and V. Khanaa (2011). Architectural patterns to design software safety based safety-critical systems. In: *Proceedings of the 2011 International Conference on Communication, Computing & Security - ICCCS '11*. ACM Press, 620.

Kyriakoulakos, Konstantinos and Dionisios N. Pnevmatikatos (2009). A novel SRAM-based FPGA architecture for efficient TMR fault tolerance support. In: *19th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE.

Laprie, J.C. et al. (1995). Architectural Issues in Software Fault Tolerance. In: *Software Fault Tolerance*. Wiley, 47–80.

Ljosland, Ingvar (2006). BUCS : Patterns and Robustness A Navigation System Case Study.

Michael, J Bret, Anil Nerode, and Duminda Wijesekera (2006). On the Provision of Safety Assurance via Safety Kernels for Modern Weapon Systems. In: *DTIC Science & Technology*, 102–105.

Miyawaki, Nii (2008). Study of Machine Safety Control. *JTEKT Engineering Journal* 1004E, 119–124.

Mutlu, Ahmet (2004). DC Motor Speed Controller Software.

Nourani, Esmaeil and Mohammad Abdollahi Azgomi (Dec. 2009). A design pattern for dependable web services using design diversity techniques and WS-BPEL. In: *2009 International Conference on Innovations in Information Technology (IIT)*. IEEE, 325–329.

Parchas, E. and R. de Lemos (2004). An architectural approach for improving availability in Web services. In: *Third Workshop on Architecting Dependable Systems (WADS)*. IET.

Preschern, Christopher (2011). PISCAS: Pisciculture Automation System Product Line. MA thesis. Graz University of Technology.

Preschern, Christopher, Nermin Kajtazovic, and Christian Kreiner (2013). Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle. In: *VikingPLoP*.

Sghairi, M. et al. (2008). Challenges in Building Fault-Tolerant Flight Control System for a Civil Aircraft. *IAENG International Journal of Computer Science* 35, 4, 495–499.

Skambraks, Martin (Sept. 2006). An Architecture for Runtime State Restoration after Transient Hardware-Faults in Redundant Real-Time Systems. In: *Conference on Emerging Technologies and Factory Automation*. IEEE, 78–85.

Solomon, Bogdan et al. (May 2007). Towards a Real-Time Reference Architecture for Autonomic Systems. In: *International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '07)*. IEEE.

Stögerer, Christoph and Wolfgang Kastner (2010). Distributed Monitoring for Component-based Traffic Management Systems. In: *Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE.

Tong, Adams N. (2007). Fabrication of deep-submicron complementary metal-oxide semiconductor devices. PhD thesis. University of Notre Dame.

Vergara-Fernandez, Antonio and Reiner Denz (2002). Reliability Analysis for the quench detection in the LHC machine. In: *8th European Particle Accelerator Conference*, 2445–2447.

Wang, Feiyi et al. (2001). SITAR : A Scalable Intrusion-Tolerant Architecture for Distributed Services. In: *Foundations of Intrusion Tolerant Systems (OASIS'03)*. June. IEEE, 5–6.

Zimmer, Marcel (2009). Prototypische Implementierung und Evaluation von Sicherheitsmustern in eingebetteten Systemen. MA thesis. Technische Universität Kaiserslautern.

Pattern-Based Safety Development Methods: Overview and Comparison

CHRISTOPHER PRESCHERN, NERMIN KAJTAZOVIC, ANDREA HÖLLER, CHRISTIAN KREINER
Institute for Technical Informatics, Graz University of Technology, Austria

Design patterns provide good solutions to re-occurring problems and several patterns and methods how to apply them have been documented for safety-critical systems. However, due to the large amount of safety-related patterns and methods, it is difficult to get an overview of their capabilities and shortcomings as there currently is no survey on safety patterns and their application methods available in literature.
To give an overview of existing pattern-based safety development methods, this paper presents existing methods from literature and discusses and compares several aspects of these methods such as the patterns and tools they use, their integration into a safety development process, or their maturity.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**] Real-Time and Embedded Systems; D.2.11 [**Software Engineering**]: Software Architecture—*Patterns*; K.6.3 [**Management of Computing and Information Systems**]: Software Management—*Software process*

ACM Reference Format:
Preschern, C. et al. 2014. Pattern-Based Safety Development Methods: Overview and Comparison. EuroPLoP 2014, 20 pages.

1. INTRODUCTION

The application of design patterns becomes increasingly attractive to specific domains as more and more domain-specific patterns and domain-specific methods of how to apply patterns come up. In particular, for safety-critical systems, which are systems whose malfunction poses a threat to human health or even human lives, the application of design patterns is very promising. That is, because the concept of applying well-proven solutions to safety problems goes well with one of the underlying ideas for constructing safety-critical systems which is to rely on approaches which proved to be successful in previous applications.

There is much literature available on safety patterns (an overview of them given in [1]) and there is much literature available on methods how to apply them like [2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12]. This makes it rather difficult on the one hand to get an overview of available safety pattern application or development methods, and on the other hand to see which of the methods described in literature are actually applicable for a specific domain, which provide appropriate tool support, or which were sufficiently applied in practice and are sufficiently mature.

This paper provides an overview of pattern-based safety development methods in literature. The presented methods were gathered with a structured literature review. The selected methods are briefly described and are then analyzed regarding several attributes such as their application domain, the type of safety patterns they apply, or regarding other attributes related to the application of the patterns or related to safety. This paper is targeted to academics as well as to practitioners who want to get an overview of pattern-based safety development methods or who want to chose an appropriate method.

This paper is structured as follows. Section 2 presents the organization of the literature review on pattern-based safety development methods. The section describes inclusion criteria for methods found in literature and describes the questions or aspects to be discussed for the methods. These aspects are covered for the 12 selected pattern-based safety development methods in Section 3 and a tabular overview of the methods and the discussed aspects is shown in Section 4. Finally, Section 5 concludes this work.

2. ORGANIZATION OF THE LITERATURE REVIEW

2.1 Selection of Pattern-Based Safety Development Methods

We were looking for papers which present a structured method to develop or modify a system architecture by applying safety patterns (e.g. methods describing how to search/retrieve/integrate safety patterns from repositories) or which perhaps even describe a whole development process based on safety patterns (e.g. methods describing how to apply safety patterns in the V-model development process). We will refer to both of the before mentioned as *pattern-based safety development methods*.

The presented methods were selected by doing a structured literature search with Google Scholar and based on the publications of any Pattern Languages of Programs (PLoP) conferences. From the Google Scholar search, the phrases *Pattern Based Safety Development*, *Safety Patterns*, *Design Patterns Safety Development*, and *Safety Pattern Application* were searched and the first 100 papers were considered. From the different PLoP conferences, all published papers were considered. Based on the considered papers, the selection of actually included papers was made by first reading the title and excluding any papers not containing the keyword *"safety"* or *"pattern"*. From the remaining papers, we read the abstracts and then decided whether to include the paper or not by subjectively judging whether the paper actually covers a pattern-based safety development method.

2.2 Questions to be answered for the Methods

The following questions will be discussed for different pattern-based safety development methods from literature to provide a good overview of these methods. The "Pattern Application" questions in this section are based on [13] and [14] which both survey methods and tools for pattern search and selection. The other questions are based on attributes explicitly promoted in some of the pattern-based safety development method papers.

Domain and Type of the Patterns

Which domain does the method primarily target?

If a method particularly says that it is targeted for a specific domain or if all publications on that method target a particular domain, then the answer to this question is that domain (e.g. "Automotive", or "Railway"). Otherwise the answer is "Safety-critical systems in general"

Which type of pattern is used by the approach?

Most safety patterns applied in different methods are quite similar and can be categorized. The answer to this question is determined if a publication explicitly states to apply some type of patterns or if all patterns applied and listed in the publications of the method are related to specific pattern types. Such pattern types could, for example, be "Architectural Patterns" or "Process Patterns".

Pattern Application

How are the patterns searched and selected?

Several methods describe guidelines which aid pattern search and selection. This question is either explicitly answered by the publications of the method or the answer is extracted from the in a publication described application of safety patterns to an example system.

How does the method support pattern integration?

Some methods provide defined pattern interfaces, descriptions, or models which help integrating and combining patterns into an existing design. The answer to this question is either explicitly stated by a publication or it is extracted from the pattern form or from the pattern application to an example system.

Which tools does the method provide?

Several methods provide tools for pattern search, instantiation, or management. Such tools are listed and described here.

Safety-Process Related Aspects

Which safety-relevant development processes or standards does the method follow?

Some of the methods themselves or the patterns are developed based on a safety standard or based on safety-related development processes. If that is the case, these standards and processes are mentioned.

Which means for safety reasoning or ensuring traceability does the method provide?

In safety, it is important to ensure that the actually implemented system meets its safety goals. Some of the methods support that by explicitly providing evidence to be used to argue for safety or by providing a structured way to link the implementation to the safety goals (traceability).

Method Maturity

How large is the research community?

Some methods are developed by a single student while others are developed by large research groups. The size of the research community to some extend reflects the effort put into an approach and its maturity. The research community size is measured by the number of publications and the number of different participating institutions. The research community is considered as *small* if all relevant publications come from a single researcher. It is considered as *medium* if more researchers are involved, and it is considered as *big* if several institutions as well as industrial partners are involved.

To which kind/size of systems has the method been applied to?

Some methods have been applied to industrial projects while others are just applied to small theoretic example systems. The answer to this question is given in all the covered methods in form of (part of) a publication describing an example for the method application.

How has the method been evaluated?

Some methods are evaluated with questionnaires, metrics, or extensive case studies. If present, the answer to this question is found in most papers at the end of the method application to an example system.

3. PATTERN-BASED SAFETY DEVELOPMENT METHODS

In this section we describe 12 pattern-based safety development methods from literature. For each method, a short description is given, the above listed questions are discussed, and one or more patterns or part of patterns are presented in a Table to give the reader a feeling what the patterns are about.

3.1 Patterns along the Safety Lifecycle

The IEC 61508 safety standard provides requirements and problems which have to be handled by safety-related systems. However, the standard does not cover many solutions for these problems. This method suggests to use patterns to connect these problems to appropriate solutions. Such patterns can, especially for newcomers, be important tools to tackle the requirements of the safety standard.

Domain and Type of the Patterns

The method itself does not provide any patterns, but provides links to other pattern collections and languages:

- [15] and [16] describes process patterns for software development
- [17], [18], [19], and [20] provide safety-related patterns to be used for system design and realization

Table 3.1 shows some examples from these pattern collections.

Table I. Example Patterns of the SaCS Method

SOFTWARE VALIDATION PLANNING [15]	The pattern contains a classical textual pattern description (including context, problem, forces, solution, consequences sections). The solution part describes the software development and validation process described in the IEC 61508 standard and provides a diagram of the activities.
FAILURE ANALYSIS [15]	Also this pattern contains a classical textual pattern description. The pattern suggests to use a systematic analysis technique for the design phase like fault tree analysis or cause consequence analysis. A diagram presenting these techniques and their relations is provided.
SEPARATED SAFETY [17]	Also this pattern contains a classical textual pattern description. The pattern suggests to separate complex controlling requirements from the safety requirements of a system and to implement the two tyes of requirements in separate systems in order to keep the part which is safety critical minimal.

Pattern Application

For the selection of appropriate patterns, the usage of the above mentioned different sets of patterns for different steps of a safety development process is suggested. The method provides references to these safety pattern collections/languages which can be looked up to be applied during different development steps. For example, during the hazard and risk analysis phase, the method suggests to look for an appropriate patterns in [15] and [16]. The referenced pattern collections/languages then provide their own pattern selection methods in order to find a specific pattern.

Safety-Process Related Aspects

The method covers the application of patterns for safety-critical systems in the context of the IEC 61508 safety standard and suggested which patterns to apply along the IEC 61508 safety development process. Some of the referenced patterns are based on or mined from the IEC 61508 standard.

Method Maturity

The method to apply safety patterns along the safety lifecycle is suggested in a single publication [3]. However, the process and architecture patterns based on the IEC 61508 standard which are recommended to be used along the safety lifecycle are developed by several different researchers or research groups.

3.2 Safe Control System Method

The Safe Control System Method (SaCS) [2] provides guidance on accepted safety engineering practices to support the conceptual design of safety systems by suggesting safety engineering practices documented as patterns during the safety development lifecycle.

Domain and Type of the Patterns

SaCS describes requirements, safety process, safety architecture, and safety assurance patterns. Some example patterns are shown in Table 3.2. Examples for safety processes described as patterns are the Fault Tree Analysis (FTA) or safety case arguments with Goal Structuring Notation (GSN). Examples for safety architecture patterns are solutions describing redundant architectures. The patterns each have a graphical icon representing them and contain detailed information about their inputs and outputs (such in-/outputs could, for example, also be documents or specifications). The patterns themselves consist of a textual description and of diagrams (depending on the kind of pattern: UML diagrams, problem frames, or safety cases). SaCS presents more than 20 (potential) patterns, but just provides detailed descriptions for 3 of them in the publications.

Table II. Example Patterns of the SaCS Method

STATE SPACE SUBSET	This is a requirement pattern. It contains textual descriptions of requirements and a graphic problem diagram. The pattern suggests to form the requirements in a way that the system realizations can be split into subsystems with overlapping and redundant functionality.
TRUSTED BACKUP	This design pattern consists of textual descriptions and a UML diagram describing its structure. The pattern suggests redundant controllers. The primary controller is accurate and effective, but not rigorously developed according to safety standards, whereas the other controller rigorously conforms to safety standards and overrides decisions from the primary controller if they don't meet safety requirements.
ADAPTING WITHIN A CONSTRAINED STATE SPACE	This safety case pattern provides a textual description and a GSN diagram which provides an argumentation for the system's safety. This argumentation is tailored for systems applying the TRUSTED BACKUP pattern.

Pattern Application

SaCS provides guidance along different safety development phases and suggests appropriate sets of patterns to be applied for each specific development step. For example, during the development phase in which a safety concept has to be established, SaCS suggests to apply the FAULT TREE ANALYSIS or FAILURE MODE AND EFFECTS ANALYSIS patterns. The pattern then provide diagrams (depending on the type of pattern: UML, problem frames, GSN, ...) to provide a starting point for applying and integrating the pattern. Besides, SaCS provides a meta-model for the patterns to define their interfaces and interactions. Thus, SaCS allows the combination of several patterns and even explicitly allows the definition of pattern compositions.

Safety-Process Related Aspects

The explicit interfaces between the patterns reaching from requirements analysis via implementation to safety assurance, enable the SaCS method to support traceability in terms of showing the connection between the requirements and their implementation. SaCS suggests using icons for all patterns. This allows one to represent the applied safety patterns in a single picture which can be seen as a kind of traceability model.

Method Maturity

There is one PhD student primarily working on the SaCS method and there are 3 publication available. [2] provide a collection of SaCS patterns and describe their meta-model, [21] apply the SaCS approach to a nuclear power plant example system, and [22] present 4 SaCS patterns and apply the method to a railway domain example. For SaCS evaluation, the publications discuss how it helps to produce a consistent, complete, correct, and reusable, conceptual design. Apart from SaCS, the overall method of representing the patterns with icons, defining their interfaces and possible compositions, and suggesting appropriate patterns during system development, is also used by the Norwegian research organization SINTEF for the security domain.

3.3 TERESA Project

TERESA (Trusted Computing Engineering for Resource Constrained Embedded Systems Applications) is a European research project which proposes an engineering method for embedded systems with focus on security and dependability (which includes safety). TERESA applies model-based techniques to build up a pool of re-usable high-quality components in form of patterns.

Domain and Type of the Patterns

Most of the patterns used in examples in the TERESA project target security; however, there are also patterns from the safety domain. There is not one fixed pool of patterns, but the TERESA approach aims to develop and document patterns during system development and perhaps also just use this pattern knowledge inside a company. All patterns follow a meta-model which ensures, for example, that the patterns have well-defined interfaces and can be combined. The pattern structure can further be defined through a specification language. The patterns consist of a domain-independent and a domain-specific model, which allows to decouple domain complexity from the core ideas of the patterns. The domain-specific patterns can contain formal annotations which can be used to validate the patterns in order to obtain trustworthy pattern.

The TERESA approach can be applied to any design pattern and focuses on the verification of the pattern purpose. For example, Table3.3 shows how the MEDIATOR pattern is formulated conforming to the TERESA pattern metamodel.

Table III. SECURE CHANNEL pattern formulated with the TERESA metamodel

SECURE CHANNEL	The pattern consists of textual descriptions (e.g. problem, solution section). The SECURE CHANNEL pattern suggests to set up an encrypted communication for sensitive data and to use unencrypted communication channels for regular data. The pattern contains a UML diagram showing its structure. Constraints for this UML diagram can be formulated with OCL and the TERESA approach supports verification of these formulations.

Pattern Application

TERESA provides several tools to define, manage, and apply patterns. The patterns are stored in a repository and there is an Eclipse-Plug-In to access the patterns from the repository during system development. The tool provides pattern search functionality and allows to integrate patterns (depending on the domain e.g. their UML diagrams) into the system. Other TERESA tools allow a developer to model constraints for the patterns, to manage and set up the pattern repository, and to model the overall development process.

Safety-Process Related Aspects

TERESA provides a general development process which instantiates patterns from the repository during different development steps. This development process can be modified to be aligned with the development process of e.g. a safety standard. For example, TERESA provides the modeled development process for the IEC 61508 or the railway safety standard. Additionally, the development process can be equipped with explicit checkpoints which require checking whether the currently developed system meets its requirements.

Method Maturity

There are 26 scientific publications and 19 project deliverable documents available at the [23]. Several of these publications and deliverables describe the TERESA development process and the patterns, other publications such as [4] and [24] describe how to align the development processes with safety standards, and some publications like [25] present the application of the TERESA approach to case studies. One deliverable evaluates the TERESA approach by defining relevant criteria such as reusability or tool support and by discussing these criteria for the application of TERESA to industrial projects. Additionally, risk estimations for these projects with and without applying the TERESA approach are compared.

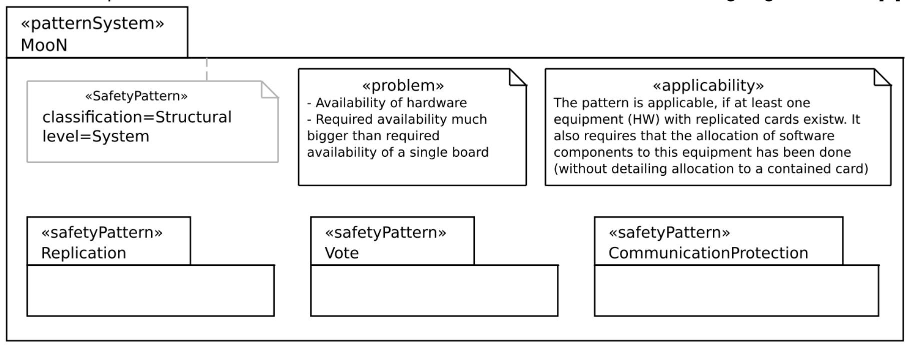
3.4 SIRSEC Project

The SIRSEC project aims to provide a platform for safety railway development and works in collaboration with the TERESA project. The focus in SIRSEC is mainly on providing a middleware, but also on using patterns during the safety development process.

Domain and Type of the Patterns

The patterns are described with the TERESA pattern meta-model. The domain-specific part of the pattern is modeled with a UML-MARTE profile to capture non-functional requirements (see Table 3.4 for an example). Each pattern contains bindings which define the interfaces of the pattern and which provide a description of how to connect them to an architecture model. The patterns also contain an informal description of the safety requirements that are addressed by a pattern.

Table IV. MooN pattern formulated with SIRSEC

MooN	The MooN M-out-of-N pattern replicates a system to N systems and votes for a consistent output of at least M systems work correctly. The pattern contains textual descriptions and a UML notation which is shown in the following figure from [5]:  «patternSystem» MooN «SafetyPattern» classification=Structural level=System «problem» - Availability of hardware - Required availability much bigger than required availability of a single board «applicability» The pattern is applicable, if at least one equipment (HW) with replicated cards existw. It also requires that the allocation of software components to this equipment has been done (without detailing allocation to a contained card) «safetyPattern» Replication «safetyPattern» Vote «safetyPattern» CommunicationProtection The UML notation shows that the pattern consists of several sub-patterns (Replication, Vote, and Communication Protection).Furthermore, the pattern contains of constraints and roles which have to be bound to an architecture during integration.

Pattern Application

Safety patterns are developed, selected, and instantiated with several TERESA tools and with an Eclipse/Papyrus tool. This Eclipse/Papyrus tool presents the patterns in different categories which a developer can choose from. When selecting a pattern, the developer chooses to which elements of an existing UML-MARTE architecture, the pattern interfaces are connected. Validation rules for the patterns then help to ensure whether the pattern was correctly instantiated and connected to the architecture model. For example, for a Triple Modular Redundancy pattern, rules can check if it is instantiated on a sufficiently high number of hardware elements.

Safety-Process Related Aspects

The patterns contain descriptions of the general safety requirements they can achieve. This provides some means of traceability.

Method Maturity

There is one major publications on the SIRSEC approach: [5] describe the application of this approach to a communication-based train control system. However, many other publication such as [4] or [26] are made in collaboration with the TERESA project.

3.5 Safety Tactic-Based Approach

The approach proposes a development process which uses tactics to build an appropriate architecture and which uses GSN to reason about the safety of the architecture.

Domain and Type of the Patterns

The approach does not actually apply patterns, but tactics. Tactics are quite similar to patterns and there is no clear definition of how to distinguish the two. Usually tactics compared to patterns are more general and abstract and cover a concept which might be part of a pattern (examples for safety tactics are given in Table 3.5). The approach presents 17 such safety tactics which are textually described.

Table V. Some examples for safety tactics

Simplicity	"Simplicity reduces the system complexity. It includes structuring methods or cutting unnecessary functionality and organizes system elements or reduces them to their core safety functionality, thus, eliminating hazards. An example for the application of the Simplicity tactic is an emergency stop switch system which is usually kept as simple as possible." [27]
Sanity Check	"The Sanity Check tactic checks whether a system state or value remains within a valid range which can be defined in the system specification or which is based on knowledge about the internal structure or nature of the system. An example for a Sanity Check is a stuck-at fault RAM-test which checks the proper functionality of the memory during system runtime. The test is based on the understanding of the memory behavior (if we write data to the memory, we should later on be able to read the same data). Faults are detected if the memory behaves differently." [27]

Pattern Application

The development process starts with a basic architecture not yet considering safety and it engineers safety into this architecture by applying safety tactics. The process and the tactics are just textually described and there is no tools support available. The process contains the following steps which are iteratively applied:

- Check each architectural element with a set of guidewords to find negative scenarios.
- Evaluate likelihood and impact of negative scenarios, rank them, and just consider the most important ones.
- Identify the architectural design space. The scenarios each contain the elements *Source*, *Stimulus*, *Environment*, and *End Effect*. For each element, a set of tactics is provided to build safety into the system.
- Choose an architectural option (safety tactic).
- Formulate a positive quality scenario which should now be fulfilled.
- Update the architecture description.

Safety-Process Related Aspects

The approach covers the integration of the above described development process into the V-model which is commonly used to development safety-critical systems. The above described development process uses scenarios and anti-scenarios. Their goals are formulated with GSN and in some cases even with GSN patterns. This provides means for safety reasoning.

Method Maturity

The Department of Computer Science at the University of York works on engineering of high integrity systems. They have several theses and other scientific publications on safety reasoning with GSN and on safety tactics and their application to case studies. For example, [28] presents the 17 safety tactics and [6] proposes the above described development process. Other safety-related development methods based on safety tactics are presented by [29], who focuses on COTS products, and by [30], who suggests an alternative process which breaks down system hazards and mitigates them by applying safety tactics. For evaluation, the feedback from stakeholders regarding the application of the method to a case study is informally discussed. Additionally a questionnaire for evaluating the method is formulated, but was not applied.

3.6 Safety Architecture Patterns

This method presents a collection of architectural safety patterns based on existing patterns from literature. The main contribution of this work is to be the first comprehensive and structured collection of these redundancy-based architecture patterns. The method primarily focuses on collecting and improving the patterns and not on discussing in detail how the patterns can be applied.

Domain and Type of the Patterns

The 15 covered patterns are (mostly redundancy-based) high-level architectural safety patterns like, for example, the N-VERSION PROGRAMMING pattern or the the TRIPLE MODULAR REDUNDANCY pattern which is shown in Table 3.6. The method presents the patterns in a uniform pattern form, provides block diagrams for the patterns, and includes safety-related metrics like reliability probabilities for the patterns. The covered patterns do not focus on a particular domain and are mostly based on patterns or techniques described in [31] and [32].

Table VI. TRIPLE MODULAR REDUNDANCY Safety Architecture Pattern

TRIPLE MODULAR REDUNDANCY	The pattern describes a system with three redundant channels. The output of the system is produced by a majority voter of the three channel outputs. Additionally to classical textual sections (e.g. problem, solution sections), the pattern also contains safety-related metrics. A metric example, is the calculation of the system reliability where the pattern contains the following formula: $R_{TMR} = R_{VOTER} * (3 * R_{CHANNEL}{}^3 - 2 * R^2_{CHANNEL})$ Regarding safety, the pattern contains recommendations for which Safety Integrity Level (SIL) the pattern should be applied. For that, the concepts which are part of the pattern and which are described in the IEC 61508 standard are listed with the pattern. The following shows these recommendations for the TRIPLE MODULAR REDUNDANCY pattern:

Techniques	**SIL1**	**SIL2**	**SIL3**	**SIL4**
Test by redundant hardware	R	R	R	R
Fault detection and diagnosis (Voter)	–	R	HR	HR

Pattern Application

The method provides a tool to access the catalog of safety patterns. The tools contains a simulation environment, which calculates and compares reliability and risk reduction values of the patterns over time depending on the hardware component failure rate. For pattern selection, the tool provides a wizard which asks questions like *"Is hardware redundancy possible?"* According to the given answers, the tool suggests one ore more patterns which are applicable. The selected pattern can then be retrieved from the tool in form of PDF document.

Safety-Process Related Aspects

The used pattern form contains information about how well the pattern solution is suited for a specific SIL level according to the IEC 61508 standard. The pattern takes this SIL level from techniques mentioned in the standard (e.g. *Hardware Redundancy*) which are applied in a specific pattern. The techniques in the standard contain recommendation for which SIL level they should be applied. The method integrates these recommendations into the patterns. Additionally, the pattern form contains reliability and risk reduction calculations which can be used during safety analysis.

Method Maturity

The main work was carried out by a single PhD student and is published in his PhD thesis [7]. The student published 6 papers on safety patterns related to the PhD thesis. Additionally, the tool for the patterns was developed as a master's thesis by [33]. There is no report of an application to an industrial project, but a very simple comparison of different applicable patterns for an example architecture is presented in [7].

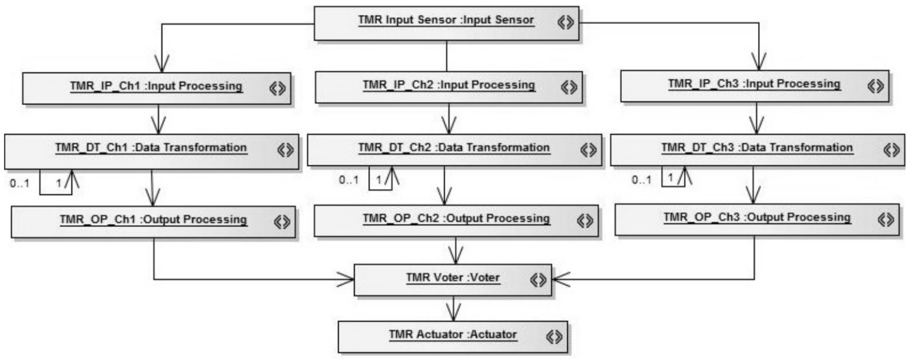
3.7 Safety Architecture Patterns + UML

The safety patterns from [7] are enhanced by including UML models and providing a structured method and tool support for UML model transformations when integrating the patterns into an existing architecture.

Domain and Type of the Patterns

The covered patterns are the 15 safety patterns from [7] (domain-independent, high-level safety architecture patterns). The patterns are extended to contain a UML diagram (as shown in Table 3.7). Architectural elements of the pattern (e.g. single channel, voter) are modeled as UML stereotypes in a UML profile. The patterns contain descriptive rules which in a general manor describe model transformations to be applied if the pattern is instantiated.

Table VII. TRIPLE MODULAR REDUNDANCY pattern including UML notation

TRIPLE MODULAR REDUNDANCY	The pattern is extended to also include the following UML notation (modified from [8]):  TMR Input Sensor :Input Sensor TMR_IP_Ch1 :Input Processing TMR_IP_Ch2 :Input Processing TMR_IP_Ch3 :Input Processing TMR_DT_Ch1 :Data Transformation TMR_DT_Ch2 :Data Transformation TMR_DT_Ch3 :Data Transformation 0..1 1 TMR_OP_Ch1 :Output Processing TMR_OP_Ch2 :Output Processing TMR_OP_Ch3 :Output Processing TMR Voter :Voter TMR Actuator :Actuator The elements of these notations are tagged and for each tag, constraints and requirements can be defined). For example, the pattern contains a requirement that all the input channels have to be connected when being instantiated. Such requirements are formally specified and can be checked after pattern instantiation.

Pattern Application

The above mentioned model transformation rules are used to integrate a pattern into an existing architecture UML model. The rules allocate pattern elements to hardware execution channels and define the flow and connection of output data between pattern elements or between the pattern and the architecture model. [34] implements a web-based repository to store and manage safety patterns. The patterns from this repository can be instantiated with a tool in form of an extension of the Sparx Systems Enterprise Architect software. This tool allows retrieving the UML class diagram of a pattern and applying the pattern's transformation rules to the existing design.

Safety-Process Related Aspects

There is no particular focus on safety standards, processes, or safety reasoning/traceability.

Method Maturity

The approach to describe the safety patterns in UML notation is described in a single publication ([8]) and there has been no other related publications by the author. The tool implementation is carried out as part of a master's thesis ([34]). There is no report of an application to an industrial project.

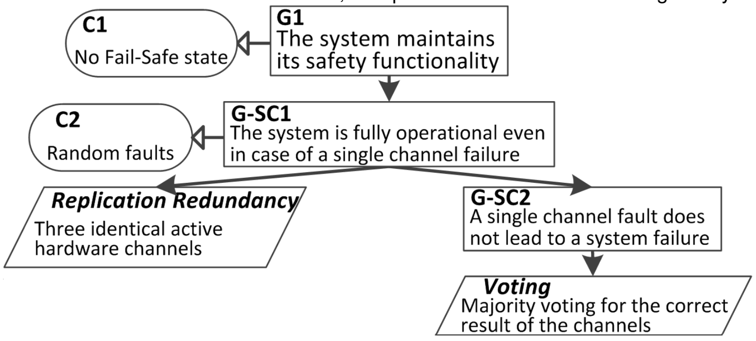
3.8 Safety Architecture Patterns + GSN

Preschern et al. extend the safety patterns from [7] to contain means for safety reasoning based on safety standards and they analyze the safety patterns for their security aspects. Furthermore, they present how to apply the patterns during safety development.

Domain and Type of the Patterns

The 15 covered patterns are based on the safety patterns from [7] (domain-independent, high-level safety architecture patterns). The patterns are presented in a uniform pattern form which additionally contains GSN diagrams for safety and security. The security GSN diagrams contain security threats which affect the system's safety and the safety GSN diagrams contain safety tactics and safety-relevant scenarios describing why the system is safe. An example pattern is shown in Table 3.8.

Table VIII. TRIPLE MODULAR REDUNDANCY pattern including GSN diagram

TRIPLE MODULAR REDUNDANCY	The pattern notation describes safety tactics of the pattern (which in case of the TRIPLE MODULAR REDUNDANCY pattern are *Redundancy* and *Voting*). Additionally, the pattern describes general scenarios relevant for the pattern. For the The TRIPLE MODULAR REDUNDANCY they are: *The system is fully operational even in case of a single channel failure.* *A single channel random fault does not lead to a system failure.* Based on the tactics and scenarios, the pattern contains the following safety GSN diagram [1]:  C1 No Fail-Safe state G1 The system maintains its safety functionality C2 Random faults G-SC1 The system is fully operational even in case of a single channel failure Replication Redundancy Three identical active hardware channels G-SC2 A single channel fault does not lead to a system failure Voting Majority voting for the correct result of the channels

Pattern Application

Preschern et al. provide a systematic analysis and presentation of the connections between the safety patterns to guide the selection of a pattern. However, there is no tool support available and the integration or composition of patterns is not discussed.

Safety-Process Related Aspects

The patterns come with GSN diagram templates which have to be developed when applying the pattern in order to provide means for safety and security reasoning. The safety GSN diagrams contain safety tactics which are applied by the pattern. These safety tactics are linked to safety methods suggested by the IEC 61508 standard. Thus, via the GSN diagrams and tactics, the patterns provide a link between the overall pattern's safety goal to actually implemented methods which are suggested by the safety standard. These methods from the standards give information on how to implement them and thus give guidance of how to implement the patterns.

Method Maturity

The main work was carried out by a single PhD student (Preschern). There are 7 publications concerning different aspects of the patterns or their application available. The main publications describing how to build the patternare are [1] and [35]. Other publications describe how the patterns are applied for the development of an industrial hydro-power plant controller ([9], [36]). To evaluate the method, security metrics for the pattern are formulated and security risk metrics before and after applying the patterns are compared.

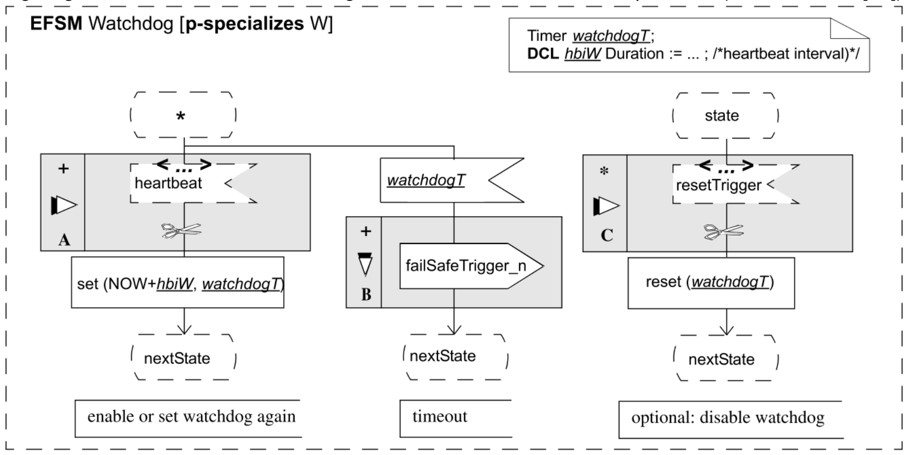
3.9 SDL Design Patterns

The method presents the application of design patterns for safety-critical distributed systems which are modeled with the Specification and Description Language (SDL) which is used to describe the communication between distributed systems and can, for example, be used to specify communication protocols.

Domain and Type of the Patterns

The presented patterns target safety-related real-time aspects of systems. Only two actual patterns are covered: The HEARTBEAT and WATCHDOG pattern (presented in Table 3.9). The patterns contain diagrams describing their structure based on an extended UML profile and they contain SDL diagrams and real-time-temporal logic models to describe their timing behavior.

Table IX. The WATCHDOG SDL pattern

WATCHDOG	The pattern contains some textual descriptions, a UML diagram describing the structure, and an SDL diagram, which is the core part of the pattern. The following figure shows the SDL diagram for the WATCHDOG pattern (taken from [10]).  EFSM Watchdog [p-specializes W] Timer watchdogT; DCL hbiW Duration := ... ; /*heartbeat interval)*/ * state + A heartbeat set (NOW+hbiW, watchdogT) nextState enable or set watchdog again watchdogT B failSafeTrigger_n nextState timeout C resetTrigger reset (watchdogT) nextState optional: disable watchdog

Pattern Application

The patterns are supposed to be applied to systems which are already modeled with SDL. The patterns contain information of how the pattern-SDL diagram has to be modified and to which elements it has to be connected when instantiating the pattern. Thus, the patterns contain structured information about pattern integration and composition; however, there is no tool support described.

Safety-Process Related Aspects

There is no particular focus on safety standards, processes. However, the patterns provide means for verification of the systems behavior. The real-time-temporal logic models of the patterns allow checking of timing requirements.

Method Maturity

[10] presents the two covered patterns and their application to a distributed airship flight control system. There are other publications focusing on the timing aspects of the approach like [37], but no other publications focus on safety aspects.

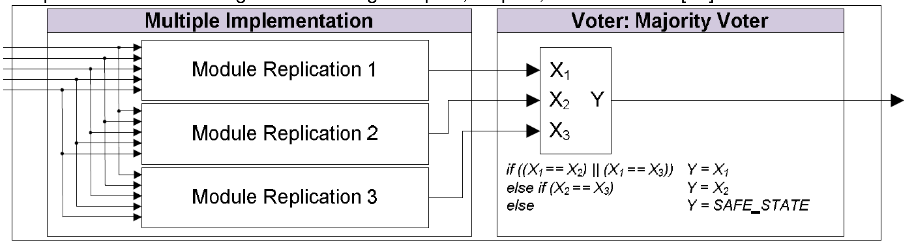
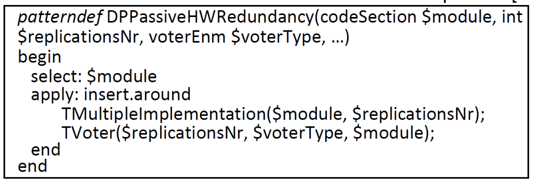
3.10 REFLECT

REFLECT (REndering FPGAs - Field-Programmable Gate Arrays - to MuLti-Core Embedded CompuTing) is a pattern-based design flow to construct FPGA systems with special focus on safety.

The Patterns

The patterns target aviation systems are are thus redundancy-based architecture patterns. The only patterns presented are the TRIPLE MODULAR REDUNDANCY and the DEFECT TOLERANCE THROUGH LOCAL SPARING patterns. The patterns contain formal transformation rules for its inputs, internal components, and outputs. Part of such a description is also shown in Table 3.10.

Table X. The TRIPLE MODULAR REDUNDANCY REFLECT pattern

TRIPLE MODULAR REDUNDANCY	The pattern contains a diagram describing its inputs, outputs, and interfaces [11]:  Multiple Implementation Module Replication 1 Module Replication 2 Module Replication 3 Voter: Majority Voter X_1 X_2 X_3 Y *if ((X₁ == X₂) || (X₁ == X₃))* $Y = X_1$ *else if (X₂ == X₃)* $Y = X_2$ *else* *Y = SAFE_STATE* The core element of the pattern, is a formal description which defines which code transformations have to take place if the pattern is applied. The following figure shows this description for the single channel elements of the TRIPLE MODLUAR REDUNDANCY pattern [11]:  *patterndef* DPPassiveHWRedundancy(codeSection $module, int $replicationsNr, voterEnm $voterType, ...) begin select: $module apply: insert.around TMultipleImplementation($module, $replicationsNr); TVoter($replicationsNr, $voterType, $module); end end

Pattern Application

REFLECT uses an aspect-oriented programming language to precisely define safety requirements of a specific system (e.g. the requirement of a maximum error probability). Additionally to the safety requirements, REFLECT assumes that a functional system is already implemented with a special notation to identify, for example, critical code sections. REFLECT provides a tool to automatically searches for the best match from a pool of safety patterns and makes automatic code transformations to integrate this pattern. For example, if a triple modular redundancy system has to be realized, REFLECT copies the critical code (which is annotated) two more times and connects the code output (which is annotated as well) to a voter component which is taken from a component library. The output of the REFLECT design flow is a new system specification which meets the safety constraints and a corresponding hardware-description-language implementation.

Safety-Process Related Aspects

The method support traceability of the specified safety requirements to their actual implementation through the patterns and through the component library.

Method Maturity

REFLECT has been brought up in [11], where is applied to a 3D path planning system for avionics systems. Two further publications ([38], [39]) describe the avionics example system and describe the design flow in more detail. There are further publications on the REFLECT design flow; however, these publications do not focus on the safety and design pattern aspects.

3.11 AltaRica Safety Patterns

The method is described as part of a research program of the company AIRBUS and formally applied redundancy-based architecture patterns to avionics systems.

Domain and Type of the Patterns

Part of the patterns are described with linear temporal logic using a formal language called AltaRica. The AltaRica pattern description is divided into several classical informal, textual pattern sections such as a problem or solution section. Furthermore, the patterns contain formal parts which describe their inputs, outputs, environment assumptions conditions, and constraints with AltaRica. In the publications just one single redundancy-based pattern (shown in Table 3.11 is fully described.

Table XI. The COLD STANDBY AltaRica pattern

COLD STANDBY	The cold standby pattern consists of two redundant channels of which one is active and the other one passive. In case of a failure in the active channel, the passive channel takes over. The pattern contains a graphical diagram showing its elements including the variables which will be used in the formal notation. The following shows this notation for the COLD STANDBY pattern [12]:
	 `sap cossap`
	 `header`
	 `problem: "the arising problem is one failure tolerance"`
	 `solution: "the proposed solution is the function passive duplication"`
	 `links: use block`
	 `redaeh`
	 `input_connection: B1.i=i1 and B2.i=i2 and B1.r=r1 and B2.r=r2;`
	 `output_connection: f1=B1.f and f2=B2.f;`
	 `output_law: The output of the SAP is always equal to the output of B1 or B2 which compose it;`

Pattern Application

Before implementing a pattern, an architecture has to be modeled with AltaRica and the method provides a tool for such modeling. The tool provides patterns in form of libraries. To implement a specific pattern, a developer has to map the input and output variables of a pattern to the architecture. After that mapping is done, attributes of the whole architecture can be checked with a model checker. The method provides a model checker for the AltaRica language. As the patterns and the architecture provide explicit interfaces, pattern composition is easily possible.

Safety-Process Related Aspects

The method suggests to also formulate the safety requirements in the formal language. If that is the case, the requirements can be traced to the actual elements and variables to which realize the requirements.

Method Maturity

There are 2 publications available on the pattern-based development with AltaRica [40], [12]. The publications are part of an Airbus research study to provide tools assisting modeling and assessment of safety architectures and also describe the application of the method to an Airbus A320 electrical system case study. As evaluation, the advantages, difficulties, and limitations of applying the method to the case study are discussed. There are many other publications on the AltaRica language, but they are not related to pattern-based approaches.

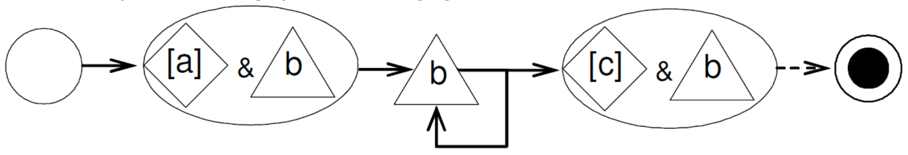
3.12 Safety Timing Templates

The method describes how to develop and verify the time behavior of safety-critical system by using a library of safety patterns which are simple formal timing-related expressions.

Domain and Type of the Patterns

The method describes timing expressions in pattern form. An example for such an expression would be that event B has to occur during event A, but not after event A. Such expressions are formulated many different notations as part of the patterns. The expressions are formulated in natural language, in a norm language (which is natural language with defined vocabulary), in linear temporal language, in computation tree logic, and in a graphical notation. However, the patterns do not contain problem or solution statements; therefore, it is debatable whether they should be considered as patterns. The method discusses that 454 such relevant logical expressions can be formulated as pattern; however, just one such expression is published in form of a full pattern (see Table 3.12).

Table XII. A safety timing template

DS-MIZ-NG-84	The patterns (or templates) do not really have names, but they have IDs containing meaningful abbreviations. For example, the MIZ part of the name means "*mit impliziten Zeitangaben*" which is German and means *"with implicit timing specification"*. The presented pattern contains the following formal specification [41]: CTL: AG (*a* → A(*b* W (*c* and *b*))) LTL: G (*a* → (*b* W (*c* and *b*))) μ-Kalkül: nu Z1.((*a* → nu Z0.((c and *b*) or (*b* and [-]Z0))) and [-]Z1) which is also presented in a graphical notation [41]:  Apart from these notations, the pattern contains information about the solution in natural language and it contains an example.

Pattern Application

A tool provides a repository for the patterns in form of a databases. The tool further helps to search and integrate the patterns into an existing formal design or into an existing set of formal requirements. The tool asks the developer questions like *"Does the Safety function contain specific timing requirements""* and shows the developer a set of patterns based on the answers. When selecting a patter, the developer has to map the inputs and outputs of the pattern to the system architecture or to the requirements.

Safety-Process Related Aspects

The method constructs a formal model regarding the timing behavior of the system. This model can be verified with a model checker and the model explicitly shows the connections from the requirements to their actual implementation.

Method Maturity

The main effort of this method was done by a single PhD student. Published work includes one PhD thesis [41], several reports from undergraduate students, and 1 scientific publication [42]. The PhD thesis describes the application of the method to an extensive example in the railway domain and discusses explicitly stated requirements for the method (formally verifyable, easy to learn and apply, ...) for the application example.

4. COMPARISON AND DISCUSSION

4.1 Pattern-Based Safety Development Methods

Table XIII shows a qualitative comparison of the presented pattern-based safety development approaches. The table shows summaries to the questions for the pattern-based safety development methods as stated in Section 2. The results in this table are also visualized in Figure 1.

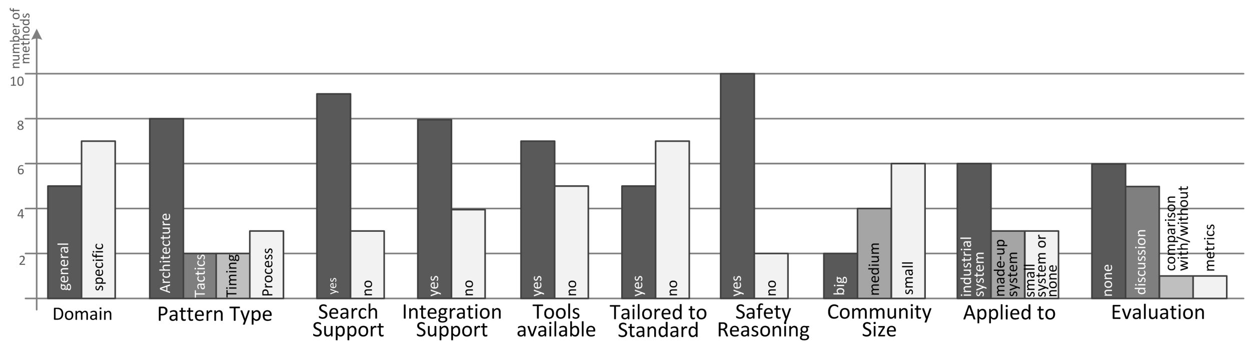


Fig. 1. Comparison of pattern-based safety-development methods

From the table we can see that some of the methods mainly focus on providing the developer with knowledge through the patterns and do not cover pattern integration aspects or safety-related aspects such as safety reasoning or traceability, while others (mainly model-based approaches) put more focus on integration and safety aspects and also provide more tool support. Related to safety aspects, some of the methods describe connections to safety standards as part of the method itself or as part of the patterns. The safety standard most commonly addressed by the methods is IEC 61508. To provide means of verification or safety reasoning, some methods produce safety evidence in form of GSN diagrams while other methods (in particular the ones focusing on timing aspects) use formal verification to prove the achievement of safety requirements.

The figure shows that most of the methods provide support regarding pattern search and integration and that most methods provide tools. Besides, most methods support safety reasoning in form of safety case or traceability support. Regarding the method maturity it is interesting to see that most methods are actually applied to industrial systems. However, most methods lack a thorough evaluation. The most common form of evaluation is to simply discuss the results of applying a method with feedback from an application example.

4.2 The used Patterns

An overview of the patterns used by the different methods is shown in Table 4.2. The table does not show all patterns mentioned by the methods, but just patterns whose full description is either published and publicly available. Some of the patterns, like for example the 52 process patterns from [15] and [16], were grouped and thus not all of the patterns are explicitly listed. From the table we can see that most of the methods apply redundancy-based architecture patterns which describe, for example, the duplication of systems in hardware or software. An example for a pattern described by most methods is the TRIPLE MODULAR REDUNDANCY system. One reason for focusing on such patterns could be that they provide well-defined descriptions in diagram form and well-defined interfaces. Thus, they can easily be integrated into existing design diagrams even with help of automated tools.

Table XIII. Comparison of Pattern-Based Safety Methods

	Domain/Type of Patterns		**Pattern Application**			**Safety-Process Aspects**		**Method Maturity**		
	Domain	**Pattern Type**	**Pattern-Search/Selection**	**Pattern Integration**	**Tool Support**	**Standards, Dev.-Process**	**Safety Reasoning / Traceability**	**Research Community**	**Applied to**	**Evaluation**
Patterns along the Safety Lifecycle	control systems	safety process and safety architecture patterns	development-phase dependent references to pattern languages	-	-	development process based on IEC 61508	some of the referenced patterns cover verification activities	small	-	-
Safe Control System Method	control systems	safety process and safety architecture patterns	shows applicable safety patterns along the development process	UML component and sequence diagrams with defined interfaces for some patterns	-	-	some patterns cover GSN; defined pattern interfaces provide link to requirements	medium	made-up system	informal general discussion
TERESA project	general safety-systems	model-based security and dependability patterns	text- or category-based pattern search tool	common pattern meta-model with defined interfaces; pattern instantiation with help of tools	pattern search/ selection/repository tool; pattern editor; property and constraint editor; process editor	meta-model to model safety-process; models present for railway-safety and IEC 61508	the process meta-model supports checkpoints to define requirement check activities	big	industrial system	discussion; risk estimations with/without TERESA
SIRSEC Project	railway safety-systems	safety architecture patterns	developer chooses from pattern categories	patterns have defined interfaces; integration with tool support	Eclipse/Papyrus tool for pattern selection and instantiation	-	patterns contain links to requirements	medium	made-up system	-
Safety Tactic-Based Approach	general safety-systems	safety tactics	tactics provided to counter elements of negative scenarios	-	-	tactic application along the safety-process discussed	GSN patterns used to build safety cases	big	industrial system	discussion on application to case studies
Safety Architecture Patterns	general safety-systems	safety architecture patterns	tool asks questions to find an appropriate pattern	-	tool to guide pattern selection and to calculate reliability and risk reduction	recommendation of the patterns for appropriate IEC 61508 SIL	-	small	small proof-of-concept system	-
Safety Architecture Patterns + UML	general safety-systems	safety architecture patterns	-	patterns with defined UML interfaces are integrated in existing architecture model	Enterprise Architect extension to manage, view, and retrieve patterns (UML)	-	-	small	-	-
Safety Architecture Patterns + GSN	general safety-systems	safety architecture patterns and safety tactics	graphical representation of pattern relationships	-	-	patterns linked to IEC 61508 methods	GSN diagrams for patterns	small	industrial system	metrics
SDL Design Patterns	distributed safety-systems	SDL patterns	-	patterns provide defined interfaces and description of how to adapt SDL diagrams	-	-	verification of timing behavior	small	industrial system	-
REFLECT	safety-related FPGAs	safety architecture patterns	automatic pattern selection according to safety requirements	code transformation based on annotated input source	code transformation tool using patterns to fulfill safety requirements	-	requirements linked to implementation	medium	made-up system	-
AltaRica Safety Patterns	avionics	safety architecture patterns	-	mapping of pattern input/output interfaces	architecture modeling and model checking tools	-	requirements linked to implementation	medium	industrial system	discussion on application to case studies
Safety Timing Templates	railway	temporal safety templates	question/answer-based guidance	mapping of pattern input/output interfaces	architecture modeling and model checking tools	-	requirements linked to implementation	small	industrial system	discussion on application to case studies

Table XIV. Patterns fully described or referenced by the pattern-based safety development methods

		Patterns along the Safety Lifecycle	Safe Control System Method	TERESA Project	SIRSEC Project	Safety Tactic-Based Approach	Safety Architecture Patterns	Safety Architecture Patterns + UML	Safety Architecture Patterns + GSN	SDL Design Patterns	REFLECT	AltaRica Safety Patterns	Safety Timing Templates
Architecture Patterns	Triple Modular Redundancy			X			X	X	X		X		
	M-out-of-N				X		X	X	X				
	Homogenous Duplex						X	X	X				
	Heterogenous Duplex						X	X	X			X	
	Monitor-Actuator						X	X	X				
	Sanity Check						X	X	X				
	Safety Executive						X	X	X				
	N-Version Programming						X	X	X				
	Recovery Block						X	X	X				
	Acceptance Voting						X	X	X				
	N-Self Checking Programming						X	X	X				
	Rec. Block & Acceptance Voting						X	X					
	Protected Single Channel						X	X	X				
	3-Level Safety Monitoring						X	X	X				
	M-out-of-N-D								X				
	Majority Voter			X									
	Data Agreement			X									
	Safe Communication Layer			X									
	Hypervisor			X									
	Reciprocal Monitoring			X									
	Local Sparing										X		
	Trusted Backup		X										
	Overall 14 architectural patterns described in [17], [18], [19], and [20]. Examples: - Separated Safety - Shared Safety Actuator - Indirect Response Check	X											
Safety Tactics	Overall 17 safety principles (safety tactics) described in [28] and [27]. Examples: - Simplicity - Condition Monitoring - Functional Redundancy - Voting					X			X				
Timing-Related	Watchdog			X			X	X	X	X			
	Heartbeat									X			
	DS-miZ-NG-84												X
Process-Related	Overall 52 process patterns described in [15], [16]. Examples: - Software Validation Planning - Failure Analysis	X											
	Adapting in Constr. State Space		X										
	State Space Subset		X										

5. CONCLUSION

This paper gave an overview of pattern-based safety development methods from literature and compared these methods regarding their patterns they use, their approach to apply the patterns, regarding safety-related aspects, and regarding their maturity. An interesting conclusion is that most of the methods apply architectural patterns and rather few methods focus on safety-related processes. Another conclusion targets the maturity of the methods. Most of them are not thoroughly evaluated and thus it is very difficult to assess their actual benefit when applying them. The reason why the methods are not thoroughly evaluated appears to be that such an evaluation requires a lot of effort and cannot easily be done in a real-life industrial project.

The presented overview and comparison on the one hand helps to see which kind of methods are available in literature, which patterns they apply, which of the methods appears to be mature, and where to get further information about the methods. On the other hand the overview and comparison shows differences between the methods and indicates their blind spots, and their capabilities. In particular, for newcomers this provides insights about which pattern-based safety development methods are available and what their benefits and drawbacks are.

ACKNOWLEDGMENTS

We would like to thank our shepherd Uwe Zdun who provided loads of valuable feedback and in particular helped significantly in making this paper more sound and valuable from a scientific point of view.

REFERENCES

[1] C. Preschern, N. Kajtazovic, and C. Kreiner, “Building a safety architecture pattern system,” in *18th European Conference on Pattern Language of Programs (EuroPLoP)*, ACM, 2013.

[2] A. A. Hauge and K. Stølen, “Developing Safe Control Systems using Patterns for Assurance,” in *The Third International Conference on Performance, Safety and Robustness in Complex Systems and Applications*, IARIA, 2013.

[3] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, “Patterns in Safety System Development,” in *PESARO 2013, The Third International Conference on Performance, Safety and Robustness in Complex Systems and Applications*, pp. 9–15, IARIA, 2013.

[4] B. Hamid, J. Geisel, A. Ziani, and D. Gonzalez, “Safety Lifecycle Development Process Modeling for Embedded Systems - Example of Railway Domain,” in *SERENE'12 Proceedings of the 4th international conference on Software Engineering for Resilient Systems*, pp. 63–75, Springer, 2012.

[5] A. Radermacher, B. Hamid, M. Fredj, and J.-L. Profizi, “S&D Patterns for Component-Based Applications with Safety Requirements,” in *European Conference on Pattern Language of Programs (EuroPLoP)*, 2013.

[6] W. Wu, *Architectural Reasoning for Safety-Critical Software Applications.* PhD thesis, University of York, 2007.

[7] A. Armoush, *Design patterns for safety-critical embedded systems.* PhD thesis, RWTH Aachen University, 2010.

[8] P. O. Antonino, T. Keuler, and P. Antonino, “Towards an Approach to Represent Safety Patterns,” in *The Seventh International Conference on Software Engineering Advances (ICSEA)*, pp. 228–237, 2012.

[9] C. Preschern, N. Kajtazovic, and C. Kreiner, “Applying and evaluating architectural iec 61508 safety patterns,” in *5th International Conference on Software Technology and Engineering (ICSTE)*, 2013.

[10] I. Fliege, A. Geraldy, R. Gotzhein, T. Kuhn, and C. Webel, “Developing safety-critical real-time systems with SDL design patterns and components,” *Computer Networks*, vol. 49, no. 5, pp. 689–706, 2005.

[11] Z. Petrov, K. Kratky, J. M. P. Cardoso, and P. C. Diniz, “Programming safety requirements in the REFLECT design flow,” in *2011 9th IEEE International Conference on Industrial Informatics*, pp. 841–847, IEEE, 2011.

[12] C. Kehren, C. Seguin, P. Bieber, C. Castel, C. Bougnol, J. Heckmann, and S. Metge, “Architecture patterns for safe design,” in *AAAF 1st Complex and Safe Systems Engineering Conference (CS2E)*, 2004.

[13] S. S. Thabasum and U. T. M. Sundar, “A Survey on Software Design Pattern Tools for Pattern Selection and Implementation,” *International Journal of Computer Science & Communication Networks*, vol. 2, no. 4, pp. 496–500, 2012.

[14] A. Birukou, “A survey of existing approaches for pattern search and selection,” in *Proceedings of the 15th European Conference on Pattern Languages of Programs - EuroPLoP '10*, ACM Press, 2010.

[15] M. Vuori, H. Virtanen, J. Koskinen, and M. Katara, “Safety Process Patterns in the Context of IEC 61508-3,” tech. rep., Tampere University of Technology, 2011.

[16] J. Koskinen, M. Vuori, and M. Katara, “Safety Process Patterns: Demystifying Safety Standards,” in *2012 IEEE International Conference on Software Science, Technology and Engineering*, pp. 63–71, IEEE, 2012.

[17] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, “Architectural patterns for functional safety,” in *Nordic Conference on Pattern Language of Programs (VikingPLoP)*, 2012.

[18] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, “Patterns for safety and control system cooperation,” in *Nordic Conference on Pattern Language of Programs (VikingPLoP)*, 2013.

[19] J. Rauhamäki, T. Vepsäläinen, and S. Kuikka, “Patterns for control system safety,” in *18th European Conference on Pattern Language of Programs (EuroPLoP)*, 2013.

[20] J. Rauhamäki and S. Kuikka, “Patterns for human aspect in safety system design,” in *Nordic Conference on Pattern Language of Programs (VikingPLoP)*, 2014.

[21] A. A. Hauge and K. Stølen, “A Pattern-Based Method for Safe Control Systems Exemplified within Nuclear Power,” in *Computer Safety, Reliability, and Security - 31st International Conference, SAFECOMP*, pp. 13–24, Springer, 2012.

[22] A. A. Hauge and K. Stølen, “SACS - A Pattern Language for Safe Adaptive Control Software,” in *Proceedings of the 18th Conference on Pattern Languages of Programs - PLoP ’11*, ACM Press, 2011.

[23] TERESA Project Website, “www.teresa-project.org.”

[24] H. Y. Zhang, B. Hamid, and D. Gouteux, “A Metamodel for Representing Safety LifeCycle Development Process,” in *The Sixth International Conference on Software Engineering Advances*, pp. 550–556, IARIA, 2011.

[25] A. Ziani, B. Hamid, and J.-M. Bruel, “A model-driven engineering framework for fault tolerance in dependable embedded systems design,” in *38th Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, 2012.

[26] J. Geisel, B. Hamid, A. Ziani, and A. Rademacher, “Pattern common modeling language for object and component architectures,” in *20th Conference on Pattern Language of Programs (PLoP)*, ACM, 2013.

[27] C. Preschern, N. Kajtazovic, and C. Kreiner, “Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle,” in *VikingPLoP*, 2013.

[28] W. Wu, “Safety Tactics for Software Architecture Design,” Master’s thesis, The University of York, 2003.

[29] A. Hill and M. Nicholson, “Safety tactics for reconfigurable process control devices,” in *4th IET International Conference on Systems Safety 2009. Incorporating the SaRS Annual Conference*, IET, 2009.

[30] Y. Huang, “Safety-Oriented Software Architecture Design Approach,” in *Proceedings of the 2013 International Conference on Information Science and Computer Applications (ISCA 2013)*, pp. 153–160, Atlantis Press, 2013.

[31] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems.* Pearson, 2002.

[32] L. Pullum, *Software fault tolerance techniques and implementation.* Artech House, 2001.

[33] A. Grinin, “Development of a catalog of design patterns for safety-critical embedded systems,” Master’s thesis, RWTH Aachen University, 2009.

[34] A. R. Olivera, “Taim : A Safety Pattern Repository,” 2012.

[35] C. Preschern, N. Kajtazovic, and C. Kreiner, “Security analysis of safety patterns,” in *20th Conference on Pattern Language of Programs (PLoP)*, ACM, 2013.

[36] C. Preschern, N. Kajtazovic, A. Höller, and C. Kreiner, “Quantitative security estimation based on safety architecture design patterns,” in *3rd International Conference on Software and Information Engineering (ICSIE)*, 2014.

[37] R. Gotzhein, “Model-driven by sdl Ű improving the quality of networked systems development,” in *Proceedings of the 7th International Conference on New Technologies of Distributed Systems (NOTERE)*, 2007.

[38] J. M. P. Cardoso, R. Nane, P. C. Diniz, Z. Petrov, K. Kratky, K. Bertels, M. Hübner, O. Goncalves, J. Gabriel, F. Coutinho, G. Constantinides, and O. W. Luk, “A new approach to control and guide the mapping of computations to fpgas,” in *in Proc. of the Intl. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2011.

[39] Z. Petrov, P. Zaykov, J. Cardoso, J. Coutinho, P. Diniz, and W. Luk, “An aspect-oriented approach for designing safety-critical systems,” in *IEEE Aerospace Conference*, 2013.

[40] P. Bieber, C. Bougnol, C. Castel, J. pierre Heckmann, C. Kehren, S. Metge, C. Seguin, P. Bieber, C. Bougnol, C. Castel, J. p. Heckmann, C. Kehren, and C. Seguin, “Safety assessment with altarica - lessons learnt based on two aircraft system studies,” in *18th IFIP World Computer Congress*, 2004.

[41] F. Bitsch, *Verfahren zur Spezifikation funktionaler Sicherheitsanforderungen für Automatisierungssysteme in Temporallogik.* PhD thesis, Universität Stuttgart, 2007.

[42] F. Bitsch, “Safety patterns - the key to formal specification of safety requirements,” in *20th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, Springer, 2001.

Lecture Notes on Software Engineering, Vol. 2, No. 1, February 2014

Applying and Evaluating Architectural IEC 61508 Safety Patterns

Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner

***Abstract*—An important step for developing a safety-critical system is the design of its architecture. The IEC 61508 standard provides a set of architectures (1oo2, 2oo3, ...) on a high level. It is left up to the system architect to refine these architectures and to come up with a set of methods which achieve the safety goals of the system. A pool of these methods is listed in part 7 of the IEC 61508 standard. However, especially for novel safety architects choosing appropriate methods and arguing for system safety through the application of these methods is rather difficult.**

In this paper we present the application of safety patterns in order to address this problem. Our safety patterns connect IEC 61508 methods to the high level architecture to provide a way to reason about a system's safety. We apply a safety pattern to two real case studies and we evaluate the pattern by comparing the IEC 61508 methods suggested by the pattern to the IEC 61508 methods actually chosen by the real system architectures.

***Index Terms*—Design patterns, functional safety, IEC 61508.**

I. Introduction

The IEC 61508 safety standard provides a set of techniques and measures which are approved and can be used to implement safety-critical systems. They are described in the part 7 of the standard and describe approaches how to control random and systematic failures. However, for system architects who are not very familiar with the standard, it is rather complicated to choose appropriate techniques and measures from the huge pool given in the standard.

To overcome this problem, we provide architectural safety patterns in [1]. These patterns can be linked to IEC 61508 techniques and measures (as we present in one of our previous works [2]) to provide an initial suggestion of which IEC 61508 techniques and measures are appropriate for a given architecture. In this paper we want to evaluate the approach of using architectural safety patterns by comparing the IEC 61508 techniques and measures suggested by the pattern with real safety-critical system architectures which apply them. We present two case studies including the information about the IEC 61508 techniques and measures they apply. Furthermore, we discuss how well the techniques and measures suggested by our safety patterns would have fitted for the two case study architectures.

Manuscript received April 30, 2013; revised August 20, 2013.

Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner are with the Institute for Technical Informatics, Graz University of Technology, Austria (e-mail: christopher.preschern@tugraz.at, nermin.kajtazovic@tugraz.at, christian.kreiner@tugraz.at).

II. IEC 61508 Safety Design Process

This section presents an introduction of how systems conforming the IEC 61508 standard are developed. Furthermore, related work on that topic is covered.

A. Development According to the IEC 61508 Standard

During the design phase of a safety-critical product, an architect first has to decide which high-level architecture is appropriate for the safety system. Part 6 of the IEC 61508 standard suggests some well-known safety architectures such as the triple modular redundancy system or other architectures with redundant hardware. With the safety pattern system in [1] we refine some of these architectures and we add some new architectures to choose from.

After choosing an appropriate architecture, a safety architect has to select safety-relevant methods to achieve the safety goals of the system. Part 7 of the IEC 61508 standards provides a pool of more than 200 techniques and measures (e.g. monitor outputs, hardware redundancy, ...) and also gives recommendations for some techniques and measures to which Safety Integrity Level (SIL) they are appropriate. The process for choosing suitable techniques and measures for an architecture is described in [3] in a pattern notation. To further guide the selection of techniques and measures for the architect, we provide a link in [2] to relate our safety design patterns witch IEC 61508 techniques and measures. According to [4] this so far missing link between safety methods and safety goals is a major shortcoming of current safety approaches.

B. Related Work

This section presents related work on software development processes which explicitly the IEC 61508 standard.

In [5], safety-related design patterns are connected to the IEC 61508 standard by giving suggestions for which patterns to use depending on the required SIL. These recommendations are based on the IEC 61508 techniques and measures which are used by the pattern. Compared to this paper, we provide a richer set of IEC 61508 techniques and measures which can be more flexibly connected to the patterns.

IEC 61508 techniques and measures are tailored to be more applicable for model-based design in the automotive domain in [6]. The techniques and measures are extended by additional information about tools and processes for model-based design. This provides a different view on the IEC 61508 techniques and measures for the automotive domain. This view is tailored to be easier understandable for automotive domain experts.

[7] maps SPICE software development processes to functional safety artifacts defined in the safety standard. The selection and verification of IEC 61508 techniques and measures is integrated in the overall SPICE safety system development process. The application of this approach is shown on several examples from the automotive domain in [8].

III. Architectural Safety Patterns

In this section we introduce safety patterns and present one of our patterns from [1] in detail. Furthermore, we show how we want to evaluate the pattern with case studies.

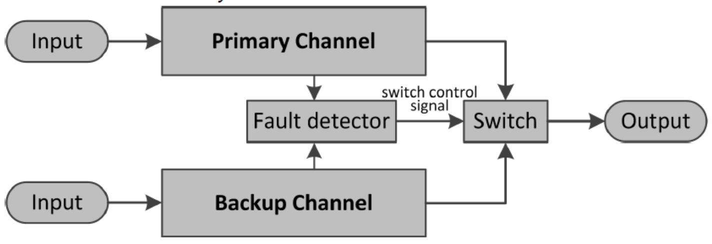
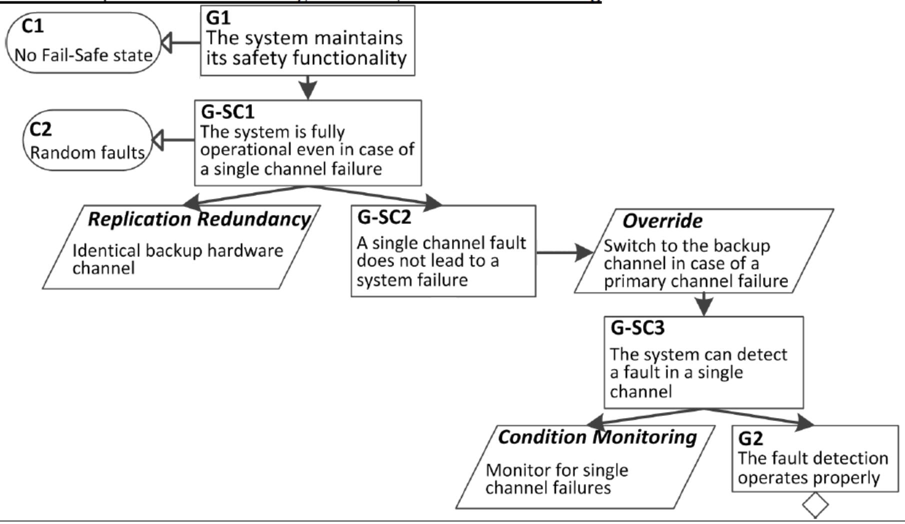
Pattern Name	HOMOGENOUS DUPLEX PATTERN	**Pattern Type**	hardware, fail-over
Also Known As	Homogeneous Redundancy Pattern, Standby-Spare Pattern, Dynamic Redundancy Pattern, Two-Channel Redundancy Pattern, 1oo2D Pattern		
Context	A safety-critical application without a fail-safe state has a high random error rate and a low systematic error rate.		
Problem	How to design a system which continues operating even in the presence of a fault in one of the system components		
Forces	- the system cannot shut down because it has no safe state - development costs should not increase - the safety standard requires high fault coverage for single-point of failure components - high availability requires hardware platforms to be maintained at the runtime		
Solution	The system consists of a *Primary Channel* (active) and a *Secondary Channel* (backup) which are two identical hardware modules. A *Fault detector* monitors the channels and controls a *Switch* to select the *Backup Channel* in case of a *Primary Channel* failure.  Input Primary Channel switch control signal Fault detector Switch Output Input Backup Channel		
GSN Diagram *(Can be used for structured safety reasoning)*	*Used Tactics:* Replication Redundancy, Override, Condition Monitoring  C1 No Fail-Safe state G1 The system maintains its safety functionality C2 Random faults G-SC1 The system is fully operational even in case of a single channel failure Replication Redundancy Identical backup hardware channel G-SC2 A single channel fault does not lead to a system failure Override Switch to the backup channel in case of a primary channel failure G-SC3 The system can detect a fault in a single channel Condition Monitoring Monitor for single channel failures G2 The fault detection operates properly		
Consequences	Systematic and random faults in a single channel are detected and masked. System reliability strongly depends on the fault coverage of the fault detection unit and on the proper functionality of the switch.		

Fig. 1. Excerpt of the homogenous duplex pattern [1].

The concept of design patterns is well known in software development. Patterns are good solutions for re-occurring problems and they discuss consequences of these solutions.

Design patterns for safety-critical hardware systems are introduced in [9], [10]. Some additional safety patterns are presented in [11] where also software-implemented safety patterns are handled. In [12] hardware as well as software patterns are collected and presented as a catalog. The catalog focuses on the consequences of the pattern application on quality attributes such as reliability, safety, cost, modifiability, and execution time. In [1], we reviewed and structured this catalog. Additionally, we linked the patterns to basic architectural design decisions, called **safety tactics**. Safety tactics are generic principles which can be applied to architectures in order to increase their safety. Safety tactics can be related to IEC 61508 techniques and measures as shown in [2], where we linked each safety tactic to a set of IEC 61508 techniques and measures by analyzing the IEC 61508 standard in a structured way.

In this paper we now combine the work of our last two papers (on a safety pattern system [1] and on connecting safety tactics to the IEC 61508 standard [2]) to obtain and evaluate safety patterns which are linked to IEC 61508 techniques and measures. These patterns can be used to structurally argue about the safety of the overall system architecture which is then linked to techniques and measures actually described by the IEC 61508 safety standard.

A. Homogenous Duplex Pattern

In this section we present one of the patterns from [1], the Homogenous Duplex Pattern. An excerpt of this pattern is shown the Fig. 1. We can see from the "*GSN Diagram*" section of the pattern, that it uses three safety tactics:

- Replication Redundancy
- Override
- Condition *Monitoring*

In [2], we linked safety tactics to IEC 61508 methods, which now allows us to connect the IEC 61508 methods to the architectural safety patterns. Table I shows the IEC 61508 techniques and measures which are related to the three tactics used by the Homogenous Duplex Pattern.

TABLE I: TACTICS AND IEC 61508 METHODS USED BY THE HOMOGENOUS DUPLEX PATTERN (BASED ON [2])

Tactics	Related IEC 61508 techniques and measures
Replication Redundancy	A.2.1 Test by redundant hardware
	A.2.5 Monitored redundancy
	A.3.5 Reciprocal comparison by software
	A.4.5 Block replication
	A.6.3 Multi-channel output
	A.6.5 Input comparison/Voting
	A.7.3 complete hardware redundancy
	A.7.5 Transmission redundancy
Override	A.1.3 Comparator
	A.1.5 Idle current principle
	A.8.1 Overvoltage protection with safety shut-off
	A.8.3 Power-down with safety shut-off
Condition Monitoring	A.1.1 Failure detection by online monitoring
	A.6.4 Monitored output
	A.8.2 Voltage control
	A.9.1 Watch dog with separate time base without time-window
	A.9.2 Watch dog with separate time base and time-window
	A.9.3 Logical monitoring of program sequence
	A.9.4 Combination of temporal and logical monitoring of program sequences
	A.9.5 Temporal monitoring with on-line check
	A.12.1 Reference sensor
	A.13.1 Monitoring

B. Pattern Evaluation

In the following we want to evaluate the link between the Homogenous Duplex Pattern and the IEC 61508 techniques and measures by comparing the techniques and measures linked to the pattern to the techniques and measures actually used in case studies which implement the pattern. In the standard, we have a big pool of techniques and measures and we want to know whether our selection from this pool by the pattern is good. This is a classical information retrieval problem; therefore, to evaluate the pattern, we calculate *Precision* and *Recall* values, which are well known information retrieval measures.

In our evaluation, the *Precision* (Equation 1) expresses how many of the techniques and measures suggested by the pattern are actually useful for the case study architecture and the *Recall* (Equation 2) expresses how many of the actually applied techniques and measures were suggested by the pattern. Additionally, we calculate the frequency of *Occurrence* (Equation 3), which tells us how many percent of the overall techniques and measures from the standard are actually applied. This measure gives us an impression of how much the effort for a safety architect is reduced if he just has to look at our suggested techniques and measures instead of the whole set from the safety standard.

$$Precision = \frac{\text{Number of methods suggested by the pattern and used by the specific architecture}}{\text{Number of methods suggested by the pattern}} \quad (1)$$

$$Recall = \frac{\text{Number of methods suggested by the pattern and used by the specific architecture}}{\text{Number of methods used by the specific architecture}} \quad (2)$$

$$Occurrence = \frac{\text{Number of methods suggested by the pattern}}{\text{Overall number of methods in the IEC 61508 standard}} \quad (3)$$

IV. CASE STUDY I: FREQUENCY CONVERTER

In this section we compare the IEC 61508 techniques and measures used by a frequency converter system to the techniques and measures described by our Homogenous Duplex Pattern.

A. System Description

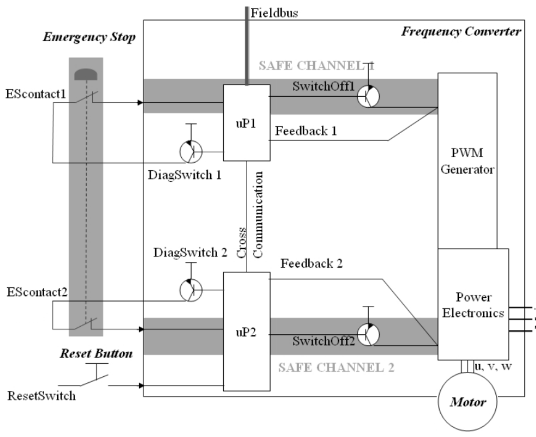


Fig. 2. Frequency converter architecture presented by Berthing *et al.* [13].

The safety-critical system we will discuss is taken from Berthing *et al*. [13]. The reason for choosing this system is that the paper was the only one we could find which gives detailed information about the applied IEC 61508 techniques and measures of an architecture. The system is a safety-related frequency converter which uses the 1oo2D architecture, which is also known as the homogenous duplex architecture. A 1oo2D architecture is a high level architecture which is described in the IEC 61508 standard. It consists of two independent hardware channels producing two outputs which are checked by a comparator. Additionally, a 1oo2D system is equipped with diagnostic tests. If the tests fail or if the outputs differ, the system goes into a fail-safe state. Fig. 2 shows the system architecture for the frequency converter presented by Berthing *et al.*

B. Evaluation of the Link of IEC 61508 Methods to the Homogenous Duplex Pattern

TABLE II: IEC 61508 TECHNIQUES AND MEASURES USED BY THE FREQUENCY CONVERTER ARCHITECTURE

Tactics	IEC 61508	Application in the frequency converter architecture
Replication Redundancy	A.2.1	Redundant electronic subsystem
	A.2.5	Monitored redundant processing units
	A.3.5	Processing units - comparison by software
	A.4.5	Processing units - ROM memory replication
	A.6.3	-
	A.6.5	Input comparison for sensors
	A.7.3	-
	A.7.5	-
Override	A.1.3	Processing unit result comparator
	A.1.5	Idle current principle for actuators
	A.8.1	Power supply unit overvoltage protection
	A.8.3	Safety shut-off for power supply unit
Condition Monitoring	A.1.1	Online monitoring of the sensors
	A.6.4	-
	A.8.2	Safety shut-off for power supply
	A.9.1	Timeout monitored by a watchdog element
	A.9.2	-
	A.9.3	Watchdog checks correct program sequence
	A.9.4	Watchdog system checks
	A.9.5	-
	A.12.1	-
	A.13.1	Monitoring of the actuators
-	A.3.2	Processing unit software self-test: walking bit
-	A.4.3	Static memory regions: one-word (8bit) signature
-	A.5.2	Walk-path RAM test for variable memory regions
-	A.7.6	Fieldbus communication data checksum
-	A.10.1	Temperature sensor monitor ventilation system
-	A.10.4	Alarm for ventilation system failures

From Table II we can see that most of the methods which are actually used by the frequency converter architecture are already suggested by the HOMOGENOUS DUPLEX PATTERN. With the information from this table, we calculate the following metrics:

- Precision=15/22=**68.2%**
- Recall=15/22=**68.2%**
- Occurrence=22/216=**10.2%**

Both, the *Precision* and the *Recall* are reasonably high to provide a good selection of suitable methods for the architecture. The *Occurrence* value shows that about 90% of the 216 techniques and measures form the IEC 61508 standard are not used by the architecture. This means that especially for inexperienced safety architects, the architectural safety patterns can be very useful during the design phase to provide an initial overview of suitable IEC 61508 for a specific system architecture.

V. CASE STUDY II: AUTOMATION SYSTEM CONTROLLER

In this section we compare the IEC 61508 techniques and measures used by an automation system controller to the techniques and measures described by our HOMOGENOUS DUPLEX PATTERN.

A. System Description

In an industrial case study we want to certify a programmable logic controller (PLC) for functional safety according to the IEC 61508 standard. The PLC system which applies the 1oo2D architecture is shown in Fig. 3. It uses redundant inputs and outputs and it has two separate hardware processing units which calculate identical tasks.

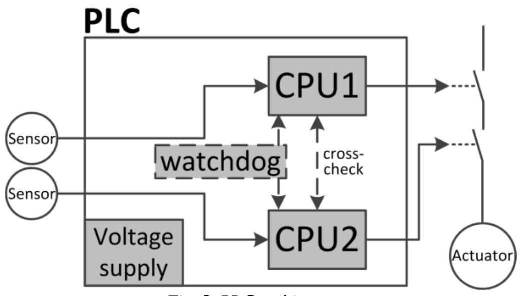


Fig. 3. PLC architecture.

B. Evaluation of the Link of IEC 61508 Methods to the Homogenous Duplex Pattern

TABLE III: IEC 61508 TECHNIQUES AND MEASURES USED BY THE PLC

Tactics	IEC 61508	Application in the frequency converter architecture
Replication Redundancy	A.2.1	✓
	A.2.5	✓
	A.3.5	✓
	A.4.5	-
	A.6.3	✓
	A.6.5	✓
	A.7.3	-
	A.7.5	✓
Override	A.1.3	-
	A.1.5	✓
	A.8.1	✓
	A.8.3	✓
Condition Monitoring	A.1.1	✓
	A.6.4	✓
	A.8.2	✓
	A.9.1	-
	A.9.2	✓
	A.9.3	✓
	A.9.4	✓
	A.9.5	✓
	A.12.1	✓
	A.13.1	-
-	A.3.2	✓
-	A.4.3	✓
-	A.5.2	✓
-	A.7.6	✓
-	A.10.1	✓
-	A.10.4	✓

Table III shows which tactics are applied by the PLC architecture. Due to confidentiality reasons we cannot explain the actual application of the IEC 61508 techniques and measures in the PLC architecture, but we can say whether it is used or not. Compared to the frequency converter case study, even more of the techniques and measures suggested by the pattern are used in the PLC architecture and we obtain the following metrics:

- Precision=17/22=**77.2%**
- Recall=17/23=**73.9%**
- Occurrence=23/216=**10.6%**

The *Precision* and *Recall* values, which are both very high. This indicates that the techniques and measures suggested by the pattern fit the actually used techniques and measures very well. The *Occurrence* is almost the same as in the frequency converter case study.

VI. Conclusion

We showed the application of an architectural safety pattern from our safety pattern system which is presented in [1]. We evaluated the suitability of the Homogenous Duplex Pattern by comparing its consequences in terms of suggested IEC 61508 methods to the methods applied in two case studies which use this architecture. From the results we can see that the patterns deliver reasonable results concerning the suggested methods.

Including the safety methods in the patterns brings benefits for system architects, because they can use the well-known pattern notation to get an overview of the IEC 61508 methods. In particular for system architects who are unfamiliar with the standard, this can be very useful. The patterns additionally provide a well-structured approach to argue about system safety by connecting the safety goals of the architecture to the IEC 61508 methods through Goal Structuring Notation.

With this paper we presented how safety pattern can be applied to design safety-critical systems and we showed how patterns can even be used to argue about the safety of a system. By showing the suitability of safety patterns for real safety architecture, we hope that this paper gives an impetus to the usage of design patterns in the safety domain.

References

[1] C. Preschern, N. Kajtazovic, and C. Kreiner, "Architectural Pattern System for Functional Safety," presented at the European Conference on Pattern Language of Programs (EuroPLoP), Irsee, Germany, July 11-15, 2013.

[2] C. Preschern, N. Kajtazovic, and C. Kreiner, "Catalog of Safety Tactics in the light of the IEC 61508 Safety Lifecycle," presented at VikingPLoP, Ikaalinen, Finland, March 21-24, 2013.

[3] M Vuori, H. Virtanen, J. Koskinen, and M. Katara, "Safety Process Patterns in the Context of IEC 61508-3," *Tech. Rep.*, Tampere University of Technology, 2011

[4] H. Becht, "Moving Towards Goal-Based Safety Management," in *Proc. Australian System Safety Conference (ASSC)*, Darlinghurst, Australia, 2011, pp. 19-26.

[5] A. Armoush and S. Kowalewski, "Safety Recommendations for Safety-Critical Design Patterns," in *Proc. Dependable Critical Systems (DDCS) in SAFECOMP*, Hamburg, Germany, 2009, pp. 9-16.

[6] M. Conrad, "Using Simulink and Real-Time Workshop Embe dded Coder for Safety-Critical Automotive Applications," presented at Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme III, TU Braunschweig, January 15-18, 2007.

[7] R. Messnarz, H. Ross, S. Habel, K. Frank, and A. Koundoussi, "Integrated Automotive SPICE and Safety Assessments," *Software process improvement and practice*, vol. 14, no. 5, pp. 179-288, 2009.

[8] R. Messnarz, I. Sokic, S. Habel, F. König, and O. Bachmann, "Extending Automotive SPICE to Cover Functional Safety Requirements and a Safety Architecture," presented at the 18th European System and Software Process Improvement and Innovation Conference (EuroSPI), Roskilde, Denmark, June 27-29, 2011.

[9] B. P. Douglass, "Safety-Critical Systems Design," *Electronic Engineering*, 1998.

[10] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, Pearson, 2002.

[11] L. Grunske, "Transformational Patterns for the Improvement of Safety Properties in Architectural Specification," presented at the Second Nordic Conference on Pattern Languages of Programs (VikingPLoP), Bergen, Norway, September 19-21, 2003.

[12] A. Armoush, "Design patterns for safety-critical embedded systems," Ph.D. dissertation, RWTH Aachen University, 2010.

[13] J. Berthing and T. Maier, "A Taxonomy for Modelling Safety-Related Architectures in Compliance with Functional Safety Requirements," in *Proc. the 26th international conference on Computer Safety, Reliability, and Security (SAFECOMP)*, Springer, Nuernburg, Germany, 2007, pp. 505-517.

Christopher Preschern received the master's degree in telematics from Graz University of Technology in 2011, focusing on software product lines, automation systems and IT-security. He is working toward the Ph.D. degree in electrical engineering at the Institute for Technical Informatics, Graz University of Technology. His research focuses on design patterns and their relationship to non-functional quality attributes, in particular safety and security.

Nermin Kajtazovic received his master's degree in telematics form Graz University of Technology in 2011. He focused on variability management in component-based architectures for the automotive domain. He is a PhD student at the Institute for Technical Informatics, Graz University of Technology. His research is related to component-based systems in the safety domain.

Christian Kreiner graduated and received the Ph.D. degree in electrical engineering from Graz University of Technology, in 1991 and 1999, respectively. From 1999 to 2007, he served as head of the R&D Department, Salomon Automation, Austria, focusing on software architecture, technologies, and processes for logistics software systems. He was in charge to establish a company-wide software product line development process and headed the product development team. There, he led and coordinated a long-term research program together with the Institute for Technical Informatics of Graz University of Technology. There, he currently leads the Industrial Informatics and Model-based Architectures group. His research interests include systems and software engineering, software technology, and process improvement.

Catalog of Security Tactics linked to Common Criteria Requirements

CHRISTOPHER PRESCHERN, Institute for Technical Informatics, Graz University of Technology

Security tactics describe security design decisions in a very general, abstract, and implementation-independent way and provide basic security design guidance. Tactics directly address system quality attributes and can be seen as building blocks for design patterns. In order to establish a more detailed security tactic collection, we link them with the Common Criteria security certification standard by establishing a connection between the security tactic goals and the Common Criteria Security Functional Requirements through Goal Structuring Notation. In this paper we give a brief introduction to the Common Criteria standard and to Goal Structuring Notation, we present the full structured and refined catalog of security tactics, and we discuss benefits of the link with the Common Criteria security standard regarding security certification.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architecture—*Patterns*; K.6.5 [**Management of computing and information systems**] Security and Protection

Additional Key Words and Phrases: security tactics, Common Criteria, Security Functional Requirements, security certification

ACM Reference Format:

Preschern, C. 2012.Catalog of Security Tactics linked to Common Criteria Requirements. Proceedings of the 2012 Conference on Pattern Languages of Programs, 17 pages.

1. INTRODUCTION

Security certification allows to evaluate a system regarding its overall security by providing a pool of requirements which have to be fulfilled. The security certification process, however, does not provide any methods to evaluate the influence of single design decisions during system development. The link between single architectural design decisions and their effect on the security quality attribute is described by [Bass et al. 2003] as security tactics. Architectural tactics, in general, address a single system quality attribute and are more general and implementation independent than design patterns, which can be composed of tactics [Kumar and Prabhakar 2010].

To evaluate the effect of architectural design decisions on security certification, a link between security tactics and the certification process is required. In this paper, we establish this link by mapping requirements given in the Common Criteria security standard to security tactics. We analyze Part 2 of the Common Critera standard which contains Security Functional Requirements (SFRs) and relate them to security tactics using Goal Structuring Notation. With this link between Common Criteria security certification and security tactics we refine security tactics. Additionally, we refine the tactics by structuring them and by gathering information from literature about their consequences and related tactics. We present the full catalog of security tactics and discuss the benefit of the established link to Common Criteria SFRs. Additionally, our security tactics catalog brings the advantage that the tactics are more structured and can provide a system architect with more detailed information about the effect of security tactics on the security quality attribute and on security certification.

Author's address: C. Preschern, Inffeldgasse 16/I, 8010 Graz, Austria; email: christopher.preschern@tugraz.at;

Section 2 of this paper presents related work on security tactics, especially on their link to security certification. Section 3 describes what architectural tactics are and gives a basic introduction to the Common Criteria security standard and to Goal Structuring Notation. In Section 4 we present the full security tactics catalog with focus on the link of the security tactics to Common Criteria requirements. In Section 5 we discuss the established link between security tactics and the Common Criteria requirements and Section 6 concludes this work.

2. RELATED WORK

Architectural tactics addressing the quality attributes availability, modifiability, performance, security, testability, and usability were introduced by [Bass et al. 2003]. This collection of tactics is extended by tactics addressing the quality attribute safety [Wu 2003] and by refinement of availability tactics [Scott and Kazman 2009]. Security tactics are extended in [Wyeth 2009], where a formal specification for them is defined. This allows to formally prove the implementation of security tactics. [Kim et al. 2009] discusses security tactics and their relationship to each other and to other tactics. They present the relationships between the tactics through feature modeling notation. An empirical study on the relationships between architectural tactics given in [Al-Daajeh et al. 2011] where the effect of safety tactics on quality attributes including security is covered.

Ryoo et al. suggest to extend security tactics by mining existing security patterns in order to find general tactics, but they do not actually extend the security tactic catalog. They also give requirements which have to hold for design decisions in order to be considered as a tactic. Tactics are domain neutral and are not attached to a particular problem, they cannot be divided into multiple tactics, and they just address a single quality attribute [Ryoo et al. 2010]. Another approach related to security patterns which uses mining is presented by Schumacher [Schumacher 2002] who suggests to use security certification standards in order to mine for security design patterns. Building patterns out of standards has the advantage, that the security standard is well accepted by a huge community and its requirements are more likely to be complete than security requirements developed by an individual. Therefore the standard allows to build patterns on a well matured basis. In [Schumacher 2003], the SFR of the Common Criteria standard are taken as input to discuss the forces affecting an architectural security pattern. Security patterns consisting of SFRs are also suggested in [Bialas 2011a] and [Bialas 2011b], where the security development process is addressed in particular. A semi-formal way to trace security tactics along the security development process is presented in [Houmb et al. 2009] where Common Criteria requirements are are modeled with UMLsec.

[Wu 2007a] analyzes Common Criteria SFRs of existing products in order to reason about the effect of these requirements on system quality attributes. He also constructs SFR requirement patterns for different domains (e.g SFR patterns suitable for operating systems). The aim of his work is to provide security system developers a good overview of relevant Common Criteria SFRs to address certain security aims. Compared to our work, Wu just focuses on the security requirements and does not describe the consequences of applying architectural tactics. We take benefit of security tactics which are a link between security quality attributes and architectural decisions. Linking the tactics to the Common Criteria SFRs allows us to extend Wu's connection between security quality attributes and SFRs to include the architectural decision which influences the quality attribute.

To further evaluate the effect of architectural decisions, such as the application of tactics, on system quality attributes, [Bass et al. 2003] suggest to construct scenarios and to evaluate different system architectures against these scenarios. In [Ellison et al. 2004] security threats are evaluated and mitigated through security tactics. Our work allows to establish a connection between the Common Criteria standard and these architecture evaluation methods by connecting SFRs to security tactics.

3. BACKGROUND

3.1 Architectural Tactics

Architectural Tactics describe basic design decisions which affect quality attributes like security for example. Compared to design patterns, tactics are related to a single quality attribute and are more general. [Ryoo et al. 2010] describe the relationship between tactics and patters as follows: *"Tactics are primitives that serve as building blocks for architectural patterns. It is also true that tactics and patterns are not at the same abstraction level. Patterns implement tactics; therefore tactics are a more abstract design concept than patterns."*

Ryoo et al. also gives basic attributes which have to be fulfilled by a design decision in order to be considered as a tactic:

—**Atomicity** - A tactic is a high level design concept and cannot be divided into other multiple tactics, but it can be refined. For example, the *Recovering From An Attack* tactic is refined by the *Restoration* and the *Identification* tactic, but it is not composed of them. Refined tactics are presented in a hierarchical tree structure.

—**Forces** - A tactic just addresses a single quality attribute and therefore does not handle trade-offs between forces.

—**Problem-Specificity** - Tactics are not specific to a problem. They are domain-neutral.

—**Completeness** - Tactics are more complete compared to design patterns in a way that less information has to be added to a tactic in order to instantiate it. This is the case because tactics are general early-design decisions.

3.2 Common Criteria

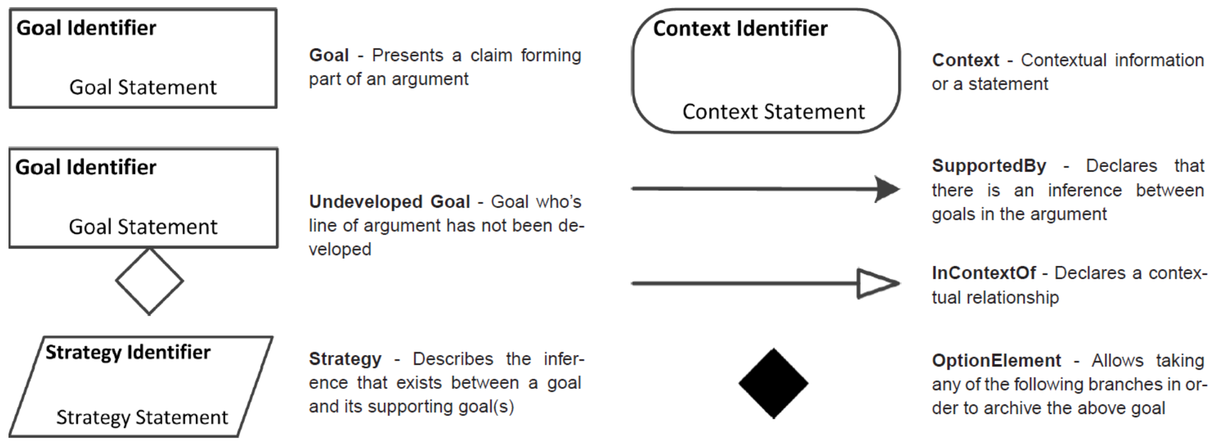
The Common Criteria is an international security standard which evolved from security standards of the Canadian, French, German, Netherlands, UK and US government. The standard is used as a basis for the evaluation of security properties and allows to compare the security of IT systems [Common Criteria Recognition Arrangement 2009]. Part 1 of the standard gives a general overview of the certification process. Part 2 contains a collection of Security Functional Requirements (SFRs) and Part 3 contains a collection of Security Assurance Requirements (SARs). Requirements are grouped into classes which are further refined into families and then further refined into components. The abbreviation of a requirement consists of three letters for the class, an underscore, three letters for the family, a dot, and the component number. The requirement for cryptographic key generation (FCS_CKM.1), for example, is the first component of the *Cryptographic Key Management* family (CKM) which is member of the *Cryptographic Support* class (FCS).

For security certification of a Target Of Evaluation (TOE), a Protection Profile has to be defined or reused. This profile is a general description of the type of TOE system and contains a set of SFRs and SARs against which the system has to be certified. If a Protection Profile already exists for the TOE domain, it can be reused. If it does not exist, it has to be constructed out of the SFRs and SARs given in the Common Criteria standard. This can be a tedious task due to the huge amount of requirements. To evaluate a TOE according to a Protection Profile, a Security Target has to be defined. This Security Target also consists of requirements for the TOE, but compared to the Protection Profile it is not general but bound to the specific implementation of the system. For the security target, a developer definitely has to create a set of Common Criteria requirements which the system has to meet. These requrements for Security Targets mostly consist of the Protection Profile requirements and probably additional requirements of the standard.

For the SARs several packages are defined which allow the developer to easily choose a suitable set of SARs. The packages are called Evaluation Assurance Levels (EALs) and represent the confidentiality one can have in the correct implementation of the system SFRs. There are, however, no packages defined for SFRs which makes it rather difficult for developers to choose an appropriate set of functional requirements. By mapping the SFRs to security tactics, this paper bridges this gap and allows developers to choose SFRs by deciding for security tactics.

3.3 Goal Structuring Notation

The Goal Structuring Notation (GSN) was developed by [Kelly and Weaver 2004] and is often used in the safety domain providing a structured way to argue for the achievement of specific goals. Recently, a standard for the GSN was published which contains the definitions of the notation and which presents approaches to use GSN in order to elaborate a specific goal [GSN Working Group 2011]. GSN was used to describe the rationale of safety tactics [Wu 2003] and several suggestions have been made to use GSN in the security domain [Kelly and Weaver 2004][Cockram and Lautieri 2007]. Figure 1 explains the GSN concepts which are used in this paper to link security tactics to Common Criteria SFRs.



Goal - Presents a claim forming part of an argument

Undeveloped Goal - Goal who's line of argument has not been developed

Strategy - Describes the inference that exists between a goal and its supporting goal(s)

Context - Contextual information or a statement

SupportedBy - Declares that there is an inference between goals in the argument

InContextOf - Declares a contextual relationship

OptionElement - Allows taking any of the following branches in order to archive the above goal

Fig. 1. GSN concepts used in this paper taken from [GSN Working Group 2011]

4. SECURITY TACTICS

In this section we give an overview of security tactics introduced by [Bass et al. 2003],we discuss the template which we use to describe the security tactics, and we present the whole refined and structured security tactic catalog.

4.1 Security Tactic Overview

Figure 2 gives an overview of the security tactics introduced by [Bass et al. 2003]. They are divided into three distinct categories. *Resisting Attacks* covers security measures which can be applied in order to prevent attacks. These tactics address the confidentiality and integrity security attributes of a system. *Detecting Attacks* and *Recovering From An Attack* aim at handling successful attacks, where *Recovering From An Attack* focuses on availability issues of a system.

4.2 Tactic Template

In [Bass et al. 2003] each of the security tactics is just described in a single paragraph. We want to structure these tactics and add additional information. We do not use the common pattern description template consisting of problem, forces, solution, and consequences to describe the security tactics, because tactics do not relate to a specific problem or context and tactics do not address trade-offs between forces [Ryoo et al. 2010]. We use the following sections to describe security tactics:

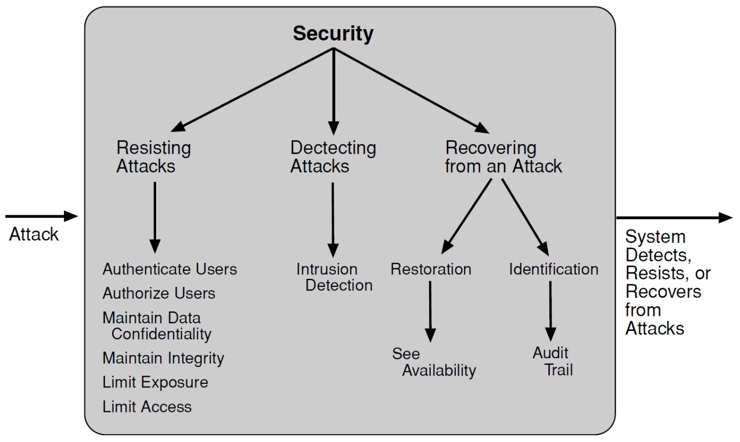


Fig. 2. Overview of security tactics [Bass et al. 2003]

—NAME - The tactic name taken from [Bass et al. 2003]

—DESCRIPTION - A general description of the tactic based on [Bass et al. 2003]

—CONSEQUENCES - This tactic section describes consequences when applying the tactic. The consequences are partially taken from patterns presented in [Hafiz et al. 2011], [Schumacher et al. 2005], and [Kienzle et al. 2002] which apply the tactics.

—RELATED TACTICS - Gives information on related tactics. The information is partially based on [Kim et al. 2009].

—KNOWN USES - Gives examples for patterns applying the tactic. The examples are taken from the security pattern catalog presented in [Hafiz et al. 2011].

—COMMON CRITERIA RATIONALE - This tactic section is the main contribution of our work. Security tactics are linked to Common Criteria v.3.1 [Common Criteria Recognition Arrangement 2009] SFRs through GSN. GSN enables us to use a structured approach to connect Common Criteria SFRs to architectural tactics. We do not break down the goal of the security tactic to a level of how it can be achieved. We break down the goal on subgoals required by the Common Criteria SFRs which can help achieving the overall goal of the security tactic. We develop each tactic by gathering and structuring Common Criteria SFRs which are related to the tactic. The selection of Common Criteria SFRs used in the GSN is based on the following sources:

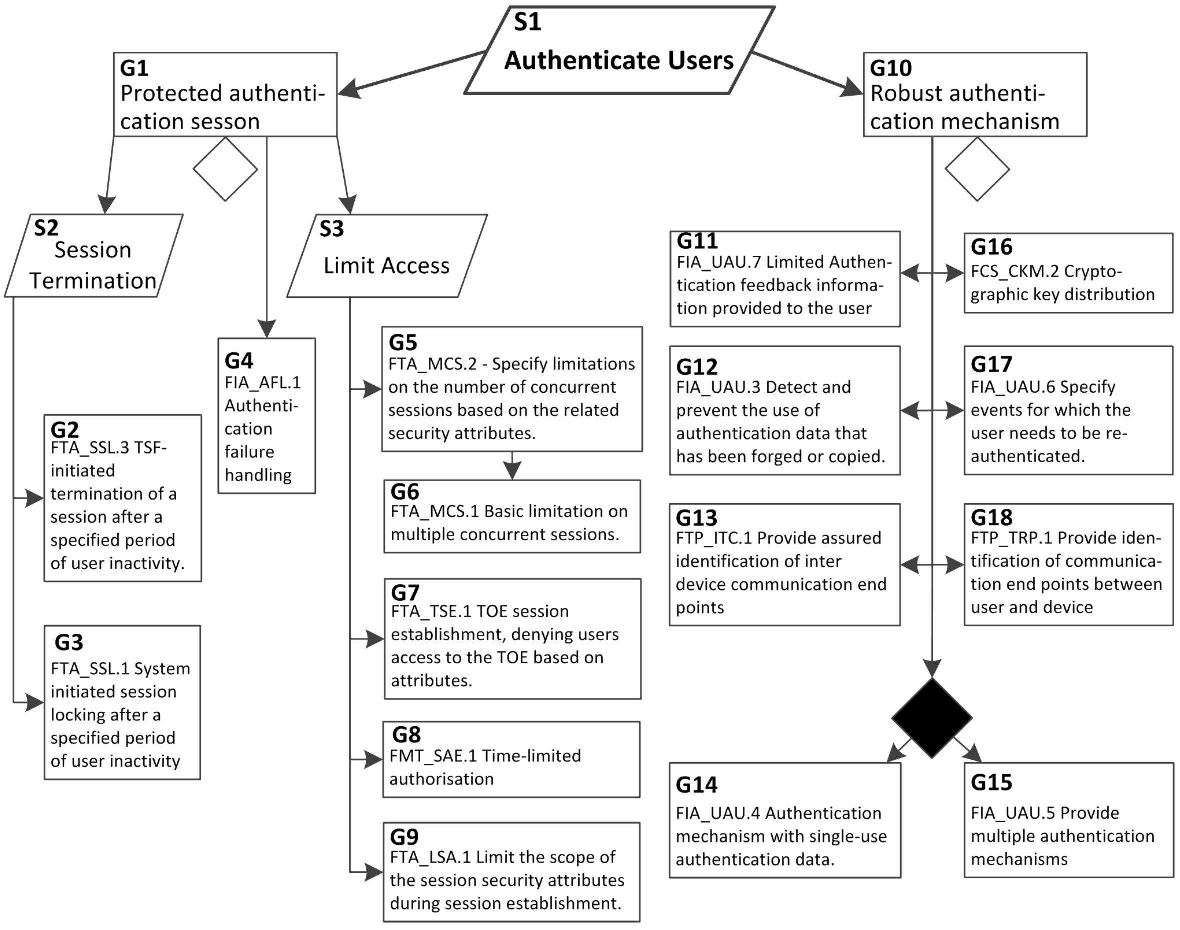
a) A thorough investigation of the Common Criteria standard Part 2
b) Protection Profiles
c) Wu's PhD thesis [Wu 2007a]

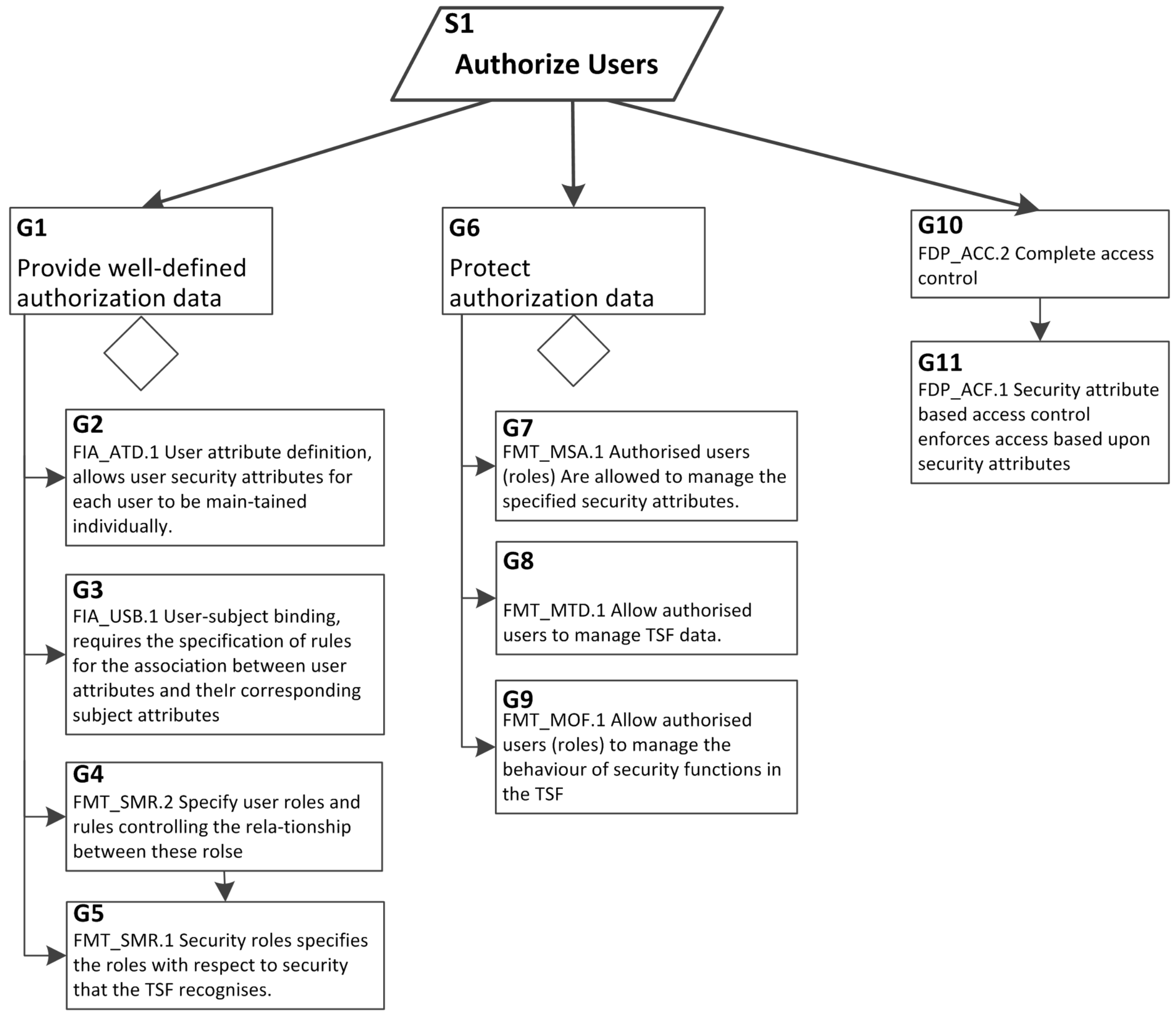
We found most of the relationships between tactics and SFRs by analyzing the SFR descriptions given in the Common Criteria standard. The SFR description of FDP_UIT (see footnote[1]), for example, suggests that this SFR is related to the *Maintain Integrity* tactic. We also analyzed approved Protection Profiles regarding their objectives and their related SFRs. The Separation Kernel Protection Profile [U.S. National Information Assurance Partnership 2007], for example, describes the objective *O.AUTHORIZED_SUBJECT*. This objective states that just authorized subjects are allowed to access restricted data. In the Protection Profile, the objective is reached if the FMT_MOF.1, FMT_MSA_EXP.1, FMT_MTD.1, and FMT_MCD_EXP.1 requirements are met. This gives us the hint that these SFRs are related to the *Authorize Users* tactic. Wu's PhD thesis [Wu 2007a] provides us

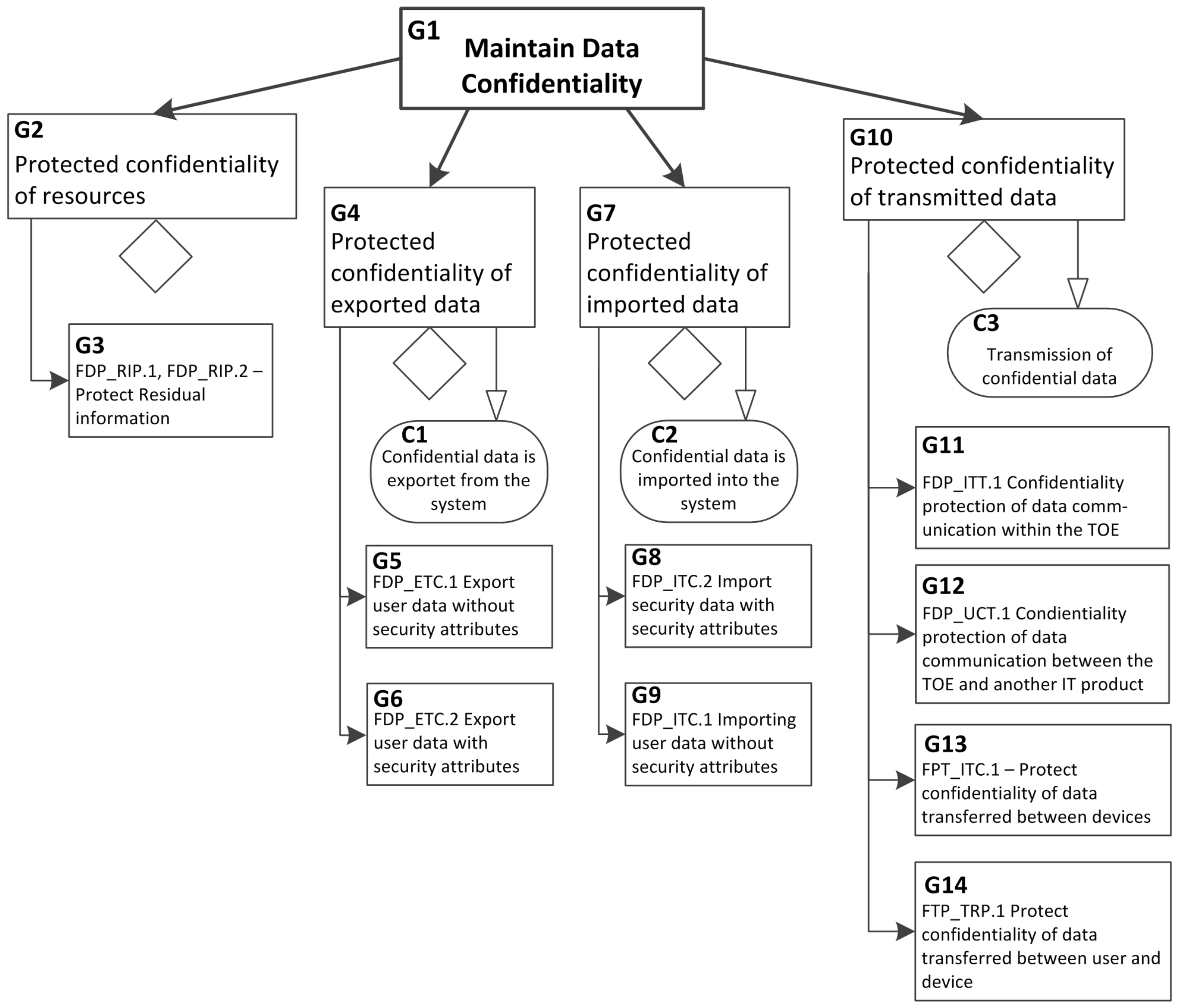
[1] FDP_UIT description: This family defines the requirements for providing integrity for user data in transit between the TOE and another trusted IT product and recovering from detectable errors. At a minimum, this family monitors the integrity of user data for modifications. Furthermore, this family supports different ways of correcting detected integrity errors. [Common Criteria Recognition Arrangement 2009]

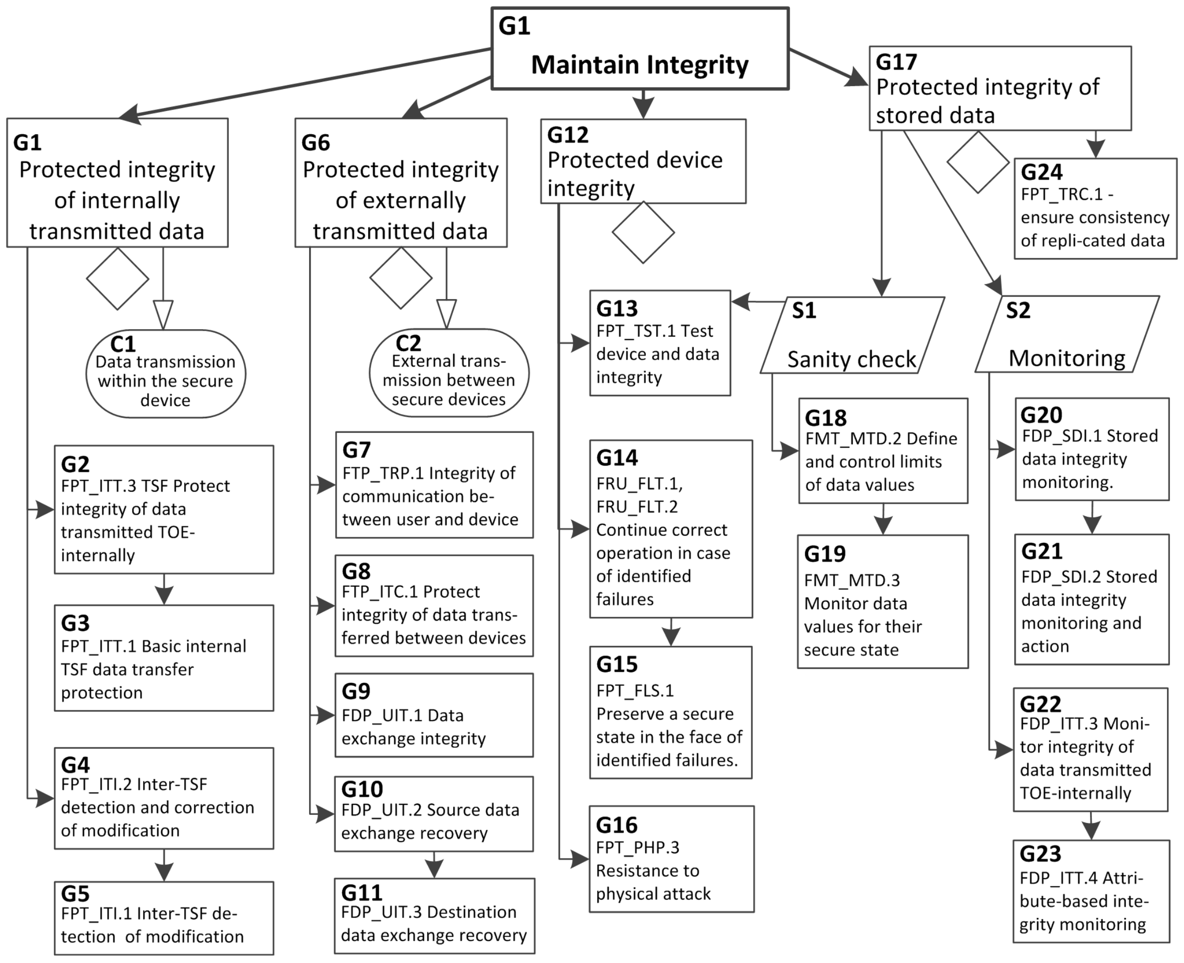
with a link between the SFRs and security attributes such as privacy or confidentiality. Some of the security tactics (for example *Maintain Data Confidentiality*) are quite close to security attributes covered by Wu. The SFRs mapped to these security attributes, therefore, can directly be included in the collection of SFRs for the corresponding tactic.
After collecting the SFRs for each tactic, we developed GSN diagrams. For each tactic, we grouped SFRs with similar aims with respect to the tactic and related them to a more general goal or strategy. These general elements or the SFRs themselves are then connected to the security tactic. Some of the tactics are represented as strategies, some of them, however, are represented as goals in the GSN. This is, because some of the security tactics are not really design decisions, but more objectives (for example: *Maintain Data Confidentiality*). The completed GSN notation for the security tactics presents a set of requirements which allows a system architect to check which requirements have to be met in order to achieve either the stated goal, or which have to be met for applying the top-level strategy (the tactic) to a system architecture. An advantage of using GSN is that the tactics can be presented in a more structured way which allows to directly use the representation for architectural reasoning. GSNs provide a good basis for architectural reasoning regarding quality attributes and have already been successfully applied to the safety domain [Wu 2007b]. The GSN diagram gives a quick overview about the goals related to a security tactic. Additionally, the SFRs can provide more detailed information on how to achieve these goals.

4.3 Tactics Catalog

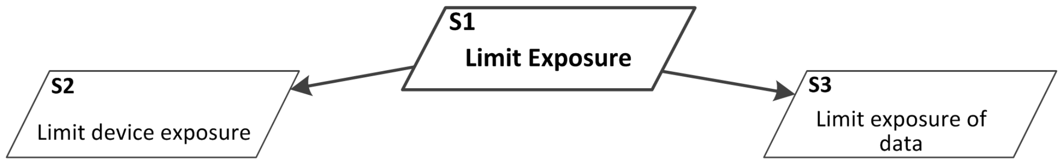
TACTIC NAME	**Authenticate Users**
DESCRIPTION	This tactic ensures that a user or computer is who he claims to be. Users/computers are authenticated by something they know, something they have, ore something they are.
COMMON CRITERIA RATIONALE	 S1 Authenticate Users G1 Protected authentication sesson G10 Robust authentication mechanism S2 Session Termination S3 Limit Access G4 FIA_AFL.1 Authentication failure handling G2 FTA_SSL.3 TSF-initiated termination of a session after a specified period of user inactivity. G3 FTA_SSL.1 System initiated session locking after a specified period of user inactivity G5 FTA_MCS.2 - Specify limitations on the number of concurrent sessions based on the related security attributes. G6 FTA_MCS.1 Basic limitation on multiple concurrent sessions. G7 FTA_TSE.1 TOE session establishment, denying users access to the TOE based on attributes. G8 FMT_SAE.1 Time-limited authorisation G9 FTA_LSA.1 Limit the scope of the session security attributes during session establishment. G11 FIA_UAU.7 Limited Authentication feedback information provided to the user G12 FIA_UAU.3 Detect and prevent the use of authentication data that has been forged or copied. G13 FTP_ITC.1 Provide assured identification of inter device communication end points G16 FCS_CKM.2 Cryptographic key distribution G17 FIA_UAU.6 Specify events for which the user needs to be re-authenticated. G18 FTP_TRP.1 Provide identification of communication end points between user and device G14 FIA_UAU.4 Authentication mechanism with single-use authentication data. G15 FIA_UAU.5 Provide multiple authentication mechanisms *Authenticate Users* is mainly based on the Common Criteria Identification and Authentication (FIA) class and on the TOE Access (FTA) class. FIA defines requirements for the authentication mechanism (G10) and FTA discusses how it can be protected (G1) by limiting the access (S3) and by maintaining session termination policies (S2).
CONSEQUENCES	Authentication mechanisms can make the access to a system more difficult and cumbersome. Authentication credentials have to be distributed/maintained
RELATED TACTICS	*Authenticate Users* is supported by the *Limit Access* tactic (S3). *Authenticate Users* is often used in combination with *Authorize Users*
KNOWN USES	Account Lockout, Assertion Builder, Authentication Enforcer, Brokered Authentication, Message Intercepting Gateway

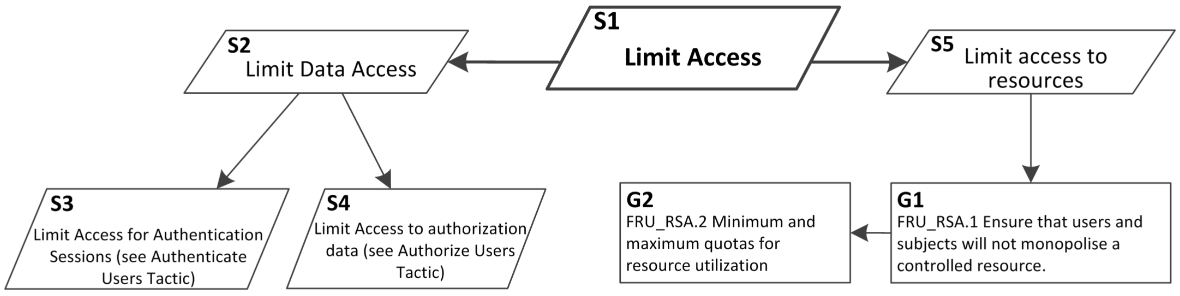
TACTIC NAME	**Authorize Users**
DESCRIPTION	This tactic ensures that only certain authenticated users have access to a resource
COMMON CRITERIA RATIONALE  S1 Authorize Users G1 Provide well-defined authorization data G2 FIA_ATD.1 User attribute definition, allows user security attributes for each user to be main-tained individually. G3 FIA_USB.1 User-subject binding, requires the specification of rules for the association between user attributes and theIr corresponding subject attributes G4 FMT_SMR.2 Specify user roles and rules controlling the rela-tionship between these rolse G5 FMT_SMR.1 Security roles specifies the roles with respect to security that the TSF recognises. G6 Protect authorization data G7 FMT_MSA.1 Authorised users (roles) Are allowed to manage the specified security attributes. G8 FMT_MTD.1 Allow authorised users to manage TSF data. G9 FMT_MOF.1 Allow authorised users (roles) to manage the behaviour of security functions in the TSF G10 FDP_ACC.2 Complete access control G11 FDP_ACF.1 Security attribute based access control enforces access based upon security attributes	Common Criteria Requirements for the *Authenticate Users* tactic mainly cover specification (G1) and protection (G6) of authorization data
CONSEQUENCES	Rules regarding authorization can easily be changed, however, possibly many rules have to be maintained. Analyzing required rules for users and understanding their implications is a rather complex task [Schumacher et al. 2005].
RELATED TACTICS	The *Authenticate Users* tactic is required as a precondition for the *Authorize Users* tactic. Defining resources a user needs authorization for follows the *Limit Exposure* tactic
KNOWN USES	Assertion Builder, Brokered Authentication, Container Managed Security, Front Door, Intercepting Web Agent, Reference Monitor, Role Based Access Control, Secure Session Object, Security Context

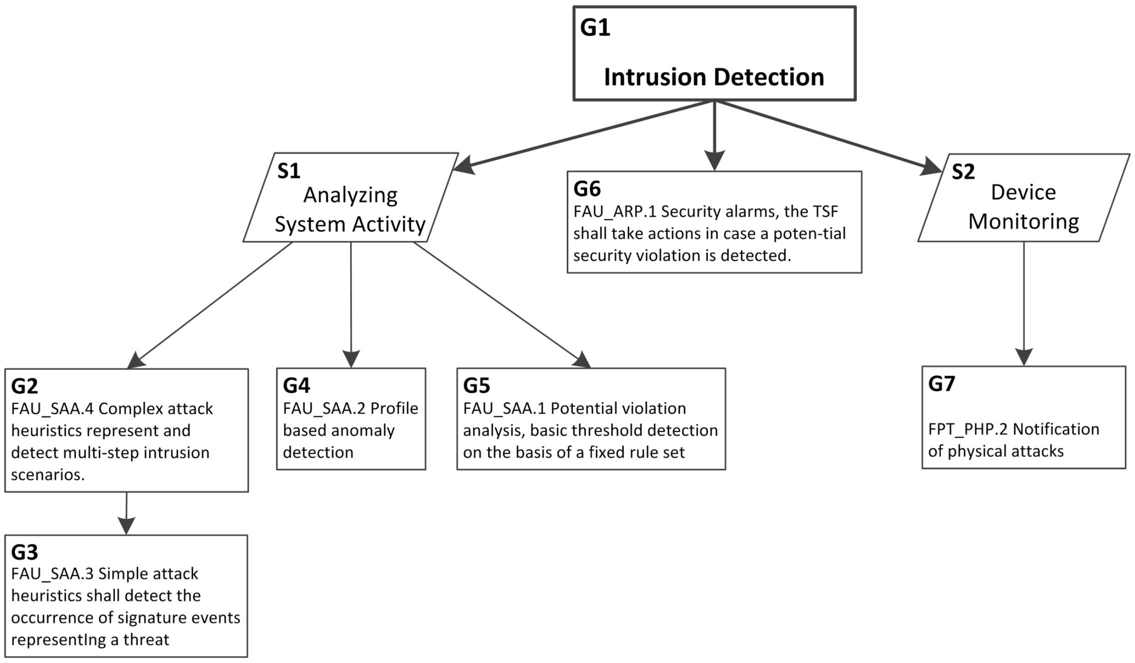
TACTIC NAME	**Maintain Data Confidentiality**
DESCRIPTION	Confidential data is protected from unauthorized access
COMMON CRITERIA RATIONALE	 G1 Maintain Data Confidentiality G2 Protected confidentiality of resources G3 FDP_RIP.1, FDP_RIP.2 – Protect Residual information G4 Protected confidentiality of exported data C1 Confidential data is exportet from the system G5 FDP_ETC.1 Export user data without security attributes G6 FDP_ETC.2 Export user data with security attributes G7 Protected confidentiality of imported data C2 Confidential data is imported into the system G8 FDP_ITC.2 Import security data with security attributes G9 FDP_ITC.1 Importing user data without security attributes G10 Protected confidentiality of transmitted data C3 Transmission of confidential data G11 FDP_ITT.1 Confidentiality protection of data comm-unication within the TOE G12 FDP_UCT.1 Condientiality protection of data communication between the TOE and another IT product G13 FPT_ITC.1 – Protect confidentiality of data transferred between devices G14 FTP_TRP.1 Protect confidentiality of data transferred between user and device Common Criteria Requirements for the *Maintain Data Confidentiality* tactic cover protection of stored (G2) and transmitted (G4) data as well as data imported (G12) into or exported (G9) from the system. *Maintain Data Confidentiality* is more a specific goal than a strategy of how to reach a goal and therefore is represented as a goal in the GSN
CONSEQUENCES	In order to enforce cryptographic protection of data, the required cryptographic secret has to be protected. Additional resources for computing and storing cryptographically protected data are required [Blakey and Heath 2004].
RELATED TACTICS	*Limit Exposure* helps to protect confidential data by eliminating possible attack vectors.
KNOWN USES	Encrypted Storage, Information Obscurity, Intercepting Web Agent, Secure Session Object

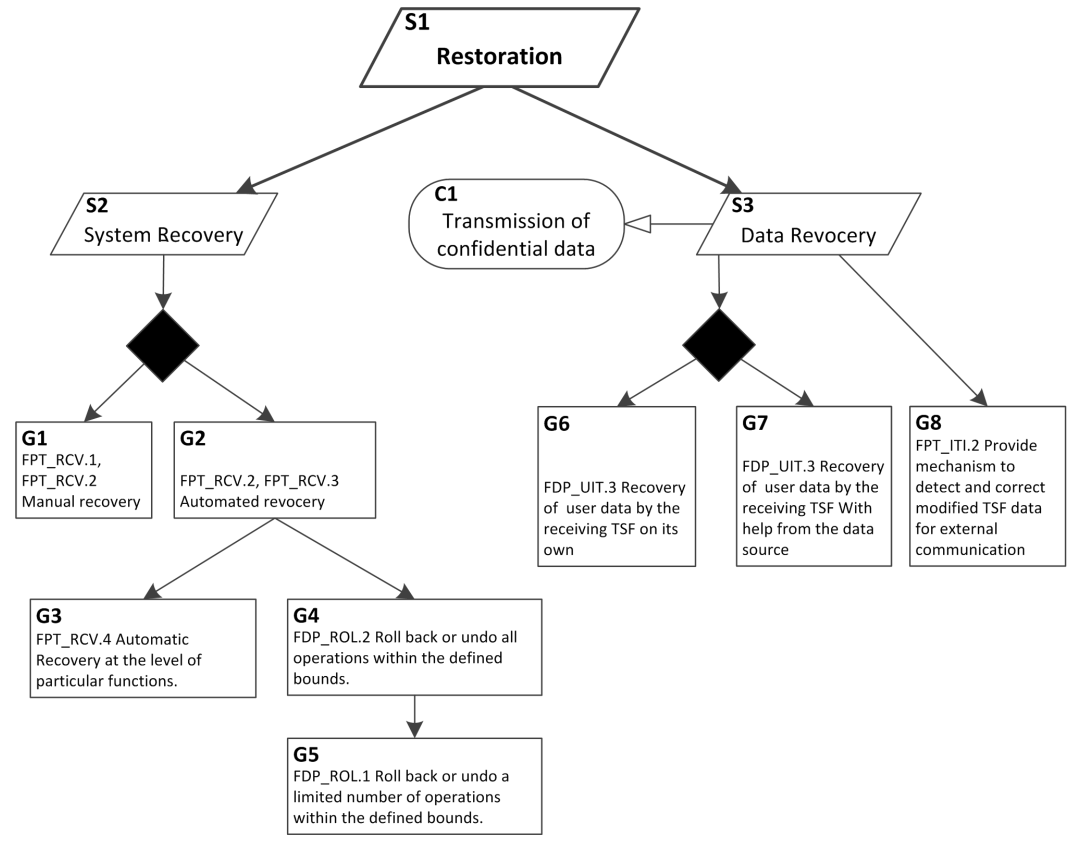
TACTIC NAME	**Maintain Integrity**
DESCRIPTION	System or data modifications should be prevented or detected
COMMON CRITERIA RATIONALE	 Most Common Criteria Requirements address the *Maintain Integrity* tactic. They can be divided in requirements protecting the device itself (G2), requirements protecting stored data (G7), and requirements protecting transmitted data (G15, G21). Most of the requirements come from the Common Criteria classes Protection of the TSF (FPT) and User Data Protection (FDP)
CONSEQUENCES	Additional resources are required to protect the integrity of data by means of cryptography [Blakey and Heath 2004]. Also any other integrity check measure requires additional resources in terms of redundant computation or storage.
RELATED TACTICS	For integrity protection the safety tactics *Sanity check* and *Monitoring* which are presented in [Wu 2003] are used. It is not surprising that safety tactics are used here, because integrity protection is part of safety measures as well [Avizienis et al. 2004].
KNOWN USES	Client Data Storage, Error Detection and Correction, Safe Data Structure

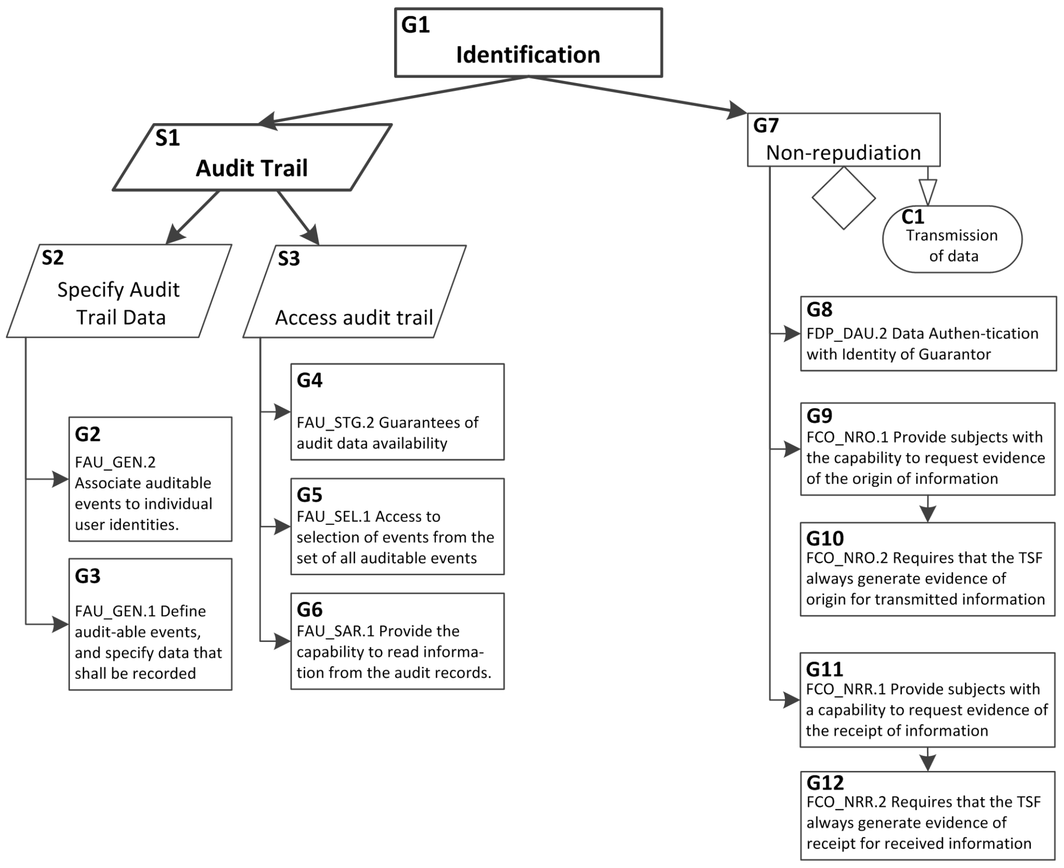
G1 Maintain Integrity
G1 Protected integrity of internally transmitted data
G6 Protected integrity of externally transmitted data
G12 Protected device integrity
G17 Protected integrity of stored data
G24 FPT_TRC.1 - ensure consistency of repli-cated data
C1 Data transmission within the secure device
C2 External trans-mission between secure devices
G13 FPT_TST.1 Test device and data integrity
S1 Sanity check
S2 Monitoring
G2 FPT_ITT.3 TSF Protect integrity of data transmitted TOE-internally
G3 FPT_ITT.1 Basic internal TSF data transfer protection
G4 FPT_ITI.2 Inter-TSF detection and correction of modification
G5 FPT_ITI.1 Inter-TSF de-tection of modification
G7 FTP_TRP.1 Integrity of communication be-tween user and device
G8 FTP_ITC.1 Protect integrity of data trans-ferred between devices
G9 FDP_UIT.1 Data exchange integrity
G10 FDP_UIT.2 Source data exchange recovery
G11 FDP_UIT.3 Destination data exchange recovery
G14 FRU_FLT.1, FRU_FLT.2 Continue correct operation in case of identified failures
G15 FPT_FLS.1 Preserve a secure state in the face of identified failures.
G16 FPT_PHP.3 Resistance to physical attack
G18 FMT_MTD.2 Define and control limits of data values
G19 FMT_MTD.3 Monitor data values for their secure state
G20 FDP_SDI.1 Stored data integrity monitoring.
G21 FDP_SDI.2 Stored data integrity monitoring and action
G22 FDP_ITT.3 Moni-tor integrity of data transmitted TOE-internally
G23 FDP_ITT.4 Attri-bute-based inte-grity monitoring

TACTIC NAME	**Limit Exposure**
DESCRIPTION	Possible attack vectors are decimated by limiting the ways security devices and data are accessible.
COMMON CRITERIA RATIONALE	 S1 Limit Exposure S2 Limit device exposure S3 Limit exposure of data No common criteria requirements directly address *Limit Exposure*. However, if *Limit Exposure* is applied to a system, less Common Criteria requirements have to be used in order to achieve system security, because less threats affect the system.
CONSEQUENCES	*Limit Exposure* is highly desirable for a system, however, it may not always be possible to enforce this tactic. *Limit Exposure* decimates possible attack vectors and therefore makes security validation easier. The tactic may however decrease system functionality.
RELATED TACTICS	*Limit Access* is a way to enforce *Limit Exposure*
KNOWN USES	Compartmentalization, Hidden Implementation, Trust Partitioning

TACTIC NAME	**Limit Access**
DESCRIPTION	Access to a resource is disabled. The user does not even have the possibility to access it [Schumacher et al. 2005].
COMMON CRITERIA RATIONALE	 S1 Limit Access S2 Limit Data Access S5 Limit access to resources S3 Limit Access for Authentication Sessions (see Authenticate Users Tactic) S4 Limit Access to authorization data (see Authorize Users Tactic) G2 FRU_RSA.2 Minimum and maximum quotas for resource utilization G1 FRU_RSA.1 Ensure that users and subjects will not monopolise a controlled resource. Common Criteria provides requirements limiting the access to data (S2) and to system resources like memory usage or processing power (S5).
CONSEQUENCES	*Limit Access* decimates possible attack vectors. However, system functionality can be affected. Administrators do not have to define access rights and enforce access rules, because access is completely denied [Schumacher et al. 2005].
RELATED TACTICS	*Limit Access* is a form of *Limit Exposure*. *Limit Access* is used for *Authenticate Users*. Also the *Authorize Users* and *Maintain Data Confidentiality* tactic inherently use *Limit Access*
KNOWN USES	Authentication Enforcer, Chroot Jail, Demilitarized Zone, Front Door, Message Intercepting Gateway, Packet Filter Firewall, Policy Enforcement Point

TACTIC NAME	**Intrusion Detection**
DESCRIPTION	Detect ongoing (intrusion detection) or past (forensic) attacks on the system
COMMON CRITERIA RATIONALE	 G1 Intrusion Detection S1 Analyzing System Activity G6 FAU_ARP.1 Security alarms, the TSF shall take actions in case a poten-tial security violation is detected. S2 Device Monitoring G2 FAU_SAA.4 Complex attack heuristics represent and detect multi-step intrusion scenarios. G4 FAU_SAA.2 Profile based anomaly detection G5 FAU_SAA.1 Potential violation analysis, basic threshold detection on the basis of a fixed rule set G7 FPT_PHP.2 Notification of physical attacks G3 FAU_SAA.3 Simple attack heuristics shall detect the occurrence of signature events representIng a threat Most *Intrusion Detection* relevant Common Criteria Requirements are found in the Security Audit (FAU) class, who's description already says that the class can be taken for intrusion detection requirements.
CONSEQUENCES	Measures allowing to identify attacks and measures taken when an attack is detected have to be specified. *Intrusion Detection* requires additional resources to log or monitor relevant data.
RELATED TACTICS	-
KNOWN USES	Dynamic Service Management

TACTIC NAME	**Restoration - Availability Tactics**
DESCRIPTION	The system can be restored after an attack
COMMON CRITERIA RATIONALE  S1 Restoration S2 System Recovery C1 Transmission of confidential data S3 Data Revocery G1 FPT_RCV.1, FPT_RCV.2 Manual recovery G2 FPT_RCV.2, FPT_RCV.3 Automated revocery G6 FDP_UIT.3 Recovery of user data by the receiving TSF on its own G7 FDP_UIT.3 Recovery of user data by the receiving TSF With help from the data source G8 FPT_ITI.2 Provide mechanism to detect and correct modified TSF data for external communication G3 FPT_RCV.4 Automatic Recovery at the level of particular functions. G4 FDP_ROL.2 Roll back or undo all operations within the defined bounds. G5 FDP_ROL.1 Roll back or undo a limited number of operations within the defined bounds.	The Common Criteria requirements cover system and data recovery with the classes Protection of the TSF (FPT) and User Data Protection (FDP)
CONSEQUENCES	Compared to simple recovery for availability, in the security domain special care has to be taken when maintaining copies of the system for *Restoration*, because this includes that multiple copies of the system can be attacked [Im and McGregor 2007].
RELATED TACTICS	*Restoration* can be in conflict with *Maintain Data Confidentiality* if multiple copies of a system have to be maintained. *Limit Access* can be applied in that case to not increase the attack surface. *Restoration* is an availability tactics and any recovery tactic presented in [Bass et al. 2003] to meet availability aims can be applied.
KNOWN USES	Checkpointed System, Error Detection and Correction, Replicated System, Standby, Tandem System

TACTIC NAME	**Identification - Audit Trail**
DESCRIPTION	Actions performed by a user are logged including an identity link
COMMON CRITERIA RATIONALE	 G1 Identification S1 Audit Trail G7 Non-repudiation C1 Transmission of data S2 Specify Audit Trail Data S3 Access audit trail G2 FAU_GEN.2 Associate auditable events to individual user identities. G3 FAU_GEN.1 Define audit-able events, and specify data that shall be recorded G4 FAU_STG.2 Guarantees of audit data availability G5 FAU_SEL.1 Access to selection of events from the set of all auditable events G6 FAU_SAR.1 Provide the capability to read informa-tion from the audit records. G8 FDP_DAU.2 Data Authen-tication with Identity of Guarantor G9 FCO_NRO.1 Provide subjects with the capability to request evidence of the origin of information G10 FCO_NRO.2 Requires that the TSF always generate evidence of origin for transmitted information G11 FCO_NRR.1 Provide subjects with a capability to request evidence of the receipt of information G12 FCO_NRR.2 Requires that the TSF always generate evidence of receipt for received information Most requirements come from the Common Criteria FAU class which explicitly addresses security audits. Other requirements come from the Communication (FCO) class and cover non-repudiation (G7) of sent or received messages. Non-repudiation is not handled as a separate security tactics in [Bass et al. 2003], but can be seen as part of the general *Identification* goal (G1).
CONSEQUENCES	Additional security relevant data for the audit trail has to be stored and protected. *Identification* possibly conflicts with user privacy requirements [Schumacher et al. 2005].
RELATED TACTICS	Maintaining an additional record of confidential audit data works against the *Limit Exposure* tactic. *Authenticate Users*, *Authorize Users*, *Maintain Data Confidentiality*, and *Maintain Integrity* are necessary to protect the audit trail.
KNOWN USES	Audit Interception, Secure Logger

Security Tactics / Common Criteria SFR Classes	Authenticate Users	Authorize Users	Maintain Data Confidentiality	Maintain Integrity	Limit Exposure	Limit Access	Intrusion Detection	Restoration Availability	Identification Audit Trail
FAU (Security Audit)							X		X
FCO (Communication)									X
FCS (Cryptographic Support)	X								
FDP (User Data Protection)	X	X	X	X				X	X
FIA (Identification&Authentication)	X	X							
FMT (Security Management)	X	X		X					
FPR (Privacy)									
FPT (Protection of the TSF)			X	X			X	X	
FRU (Resource Utilisation)				X		X			
FTA (TOE Access)	X								
FTP (Trusted Path/Channels)	X		X	X					

Table I: Mapping of the Common Criteria SFR classes to security tactics

5. DISCUSSION

Table I shows that no Common Criteria SFRs could be found which directly address the *Limit exposure* tactic. However, *Limit exposure* is a valid security tactic. It is a basic security principle and is often applied in security patterns. *Limit Exposure* fulfills all necessary requirements for tactics (atomicity, forces, problem-specificity, completeness) which are described in [Ryoo et al. 2010]. However, *Limit Exposure* has no direct functional aim and therefore is not directly addressed by the SFRs. This illustrates that the SFRs can be taken to enhance the tactics, but they cannot be taken as the single basis for security tactics.

The security tactics also do not address all aspects of security. Privacy, for example, is not handled at all. The whole Common Criteria class addressing privacy (FPR) is not mapped to any of the security tactics. Apart from this class, at least parts of all other SFR classes were mapped to at least one security tactic. This shows us that the security tactics appear to be incomplete, because they do not address all security quality attributes.

Most of the SFRs are mapped to the security tactics addressing authorization, authentication, confidentiality, and integrity. This indicates that these tactics are especially well suited to be further refined into sub-tactics, which could be based on the presented strategies in the goal structuring notation of the corresponding tactic. Confidentiality, for example, can be seen as a separate quality attribute but it is just addressed by the *Maintain Data Confidentiality* tactic. Also the integrity quality attribute is just addressed by one security tactic, *Maintain Integrity*. These two tactics are mapped to many SFRs which suggests that these two security tactics should be further refined. Another explanation for the imbalanced SFR distribution is that some security tactics rather address system requirements than architectural design decisions. *Maintain integrity*, for example just says that the quality attribute integrity has to be met and gives no design decision on how to achieve that. This is the reason why we represented the *Maintain Integrity* tactic as a goal in the GSN. This indicates that the tactic is not very well chosen. Another indicator for this is that well known security principles such as defense in depth or the least privilege principle cannot be found in the security tactics catalog. Therefore, we think that the security tactics catalog is rather incomplete and we leave it up to future work to revisit security tactics regarding their structure and completeness by adding or modifying the existing tactics.

6. CONCLUSIONS

In this paper we present a full catalog of security tactics with focus on the link between the tactics and the Common Criteria security standard. Moreover, this catalog provides more detailed and structured descriptions of security tactics compared to the initially presented tactics by [Bass et al. 2003].

The link between security tactics and SFRs allows easier evaluation of the influence of SFRs on system architectures. Security architects who have to certify their products can benefit from this link in three ways:

—Common Criteria *Protection Profile* and *Security Target* designers can construct a security architecture by using security tactics and can then see which SFRs are appropriate for the architecture. This can reduce the design effort by giving an initial advice on the SFRs which should be included for the system certification.

—Common Criteria *Target Of Evaluation* designers have a given set of SFRs which have to be achieved. The catalog suggests them a set of architectural security tactics which can achieve these SFRs.

—Common Criteria *Target of Evaluation* designers can use the catalog to argue whether their architecture uses a specific SFR and can relate this SFR to high level architectural design decisions by using the GSN diagrams.

Security architects who do not have to certify their products can still benefit from the catalog. If they want to apply a security tactic, they just have to look at the Common Criteria descriptions of the SFRs which are connected to the security tactic to get a basic idea how the tactic can be implemented. Another application of our refined security tactics catalog could be to enhance the information of existing security patterns which use the tactics. [Kumar and Prabhakar 2010] explains how patterns can be decomposed into their basic underlying tactics. The consequences section of security patterns could be extended with information about the effect on security certification by decomposing the pattern into its security tactics.

We believe that the presented collection of security tactics encourages the usage of security tactics and security patterns for products which have to be Common Criteria certified and provides a possibility to structurally argue about the usage of SFRs in high level design decisions.

ACKNOWLEDGMENTS

I would like to thank my shepherd Kiran Kumar very much for significantly improving the structure and the content of this paper by giving helpful hints and comments during the shepherding process.

REFERENCES

AL-DAAJEH, S. H., AL-QTAISH, R. E., AND AL-QIREM, F. 2011. Engineering Dependability to Embedded Systems Software via Tactics. *International Journal of Software Engineering and Its Application 5,* 4, 45–62.

AVIZIENIS, A., LAPRIE, J.-C., RANDELL, B., AND LANDWEHR, C. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing 1,* 1, 11–33.

BASS, L., CLEMENTS, P., AND KAZMAN, R. 2003. *Software Architecture in Practice* 2nd Ed. Addison-Wesley, Reading, Massachusetts.

BIALAS, A. 2011a. Common Criteria Related Security Design Patterns for Intelligent Sensors - Knowledge Engineering-Based Implementation. *Sensors (Basel, Switzerland) 11,* 8, 85–114.

BIALAS, A. 2011b. Patterns Improving the Common Criteria Compliant IT Security Development Process. In *Dependable Computer Systems (Advances in Intelligent and Soft Computing)* Volume 97 Ed. Springer-Verlag, 1–16.

BLAKEY, B. AND HEATH, C. 2004. Security Design Patterns. Tech. rep., The Open Group.

COCKRAM, T. J. AND LAUTIERI, S. R. 2007. Combining Security and Safety Principle in Practice. In *2nd Institution of Engineering and Technology International Conference on System Safety*. IEEE, 159–164.

COMMON CRITERIA RECOGNITION ARRANGEMENT. 2009. Common Criteria Standard v3.1. http://www.commoncriteriaportal.org/cc/.

ELLISON, R. J., MOORE, A. P., AND KLEIN, M. 2004. Security and Survivability Reasoning Frameworks and Architectural Design Tactics. Tech. Rep. September, Software Engineering Institute Carnegie Mellon.

GSN WORKING GROUP. 2011. GSN Community Standard Version 1. http://www.goalstructuringnotation.info/.

HAFIZ, M., ADAMCZYK, P., AND JOHNSON, R. 2011. Growing a Pattern Language (for Security). In *18th Conference on Pattern Languages of Programs*.

HOUMB, S. H., ISLAM, S., KNAUSS, E., JÜRJENS, J., AND SCHNEIDER, K. 2009. Eliciting security requirements and tracing them to design: an integration of Common Criteria, heuristics, and UMLsec. *Requirements Engineering 15,* 1, 63–93.

IM, T. AND MCGREGOR, J. 2007. Security in the Context of Dependability. In *Cyber Security and Information Infrastructure Workshop*.

KELLY, T. AND WEAVER, R. 2004. The Goal Structuring Notation Ű A Safety Argument Notation. In *Proceedings of the Dependable Systems and Networks Conference*.

KIENZLE, D. M., ELDER, M. C., TYREE, D., AND EDARDS-HEWITT, J. 2002. Security Patterns Repository. Tech. rep.

KIM, S., KIM, D.-K., LU, L., AND PARK, S. 2009. Quality-driven architecture development using architectural tactics. *Journal of Systems and Software 82,* 8, 1211–1231.

KUMAR, K. AND PRABHAKAR, T. V. 2010. Pattern-oriented Knowledge Model for Architecture Design. In *17th Conference on Pattern Languages of Programs*.

RYOO, J., LAPLANTE, P., AND KAZMAN, R. 2010. A Methodology for Mining Security Tactics from Security Patterns. In *2010 43rd Hawaii International Conference on System Sciences*. IEEE, 1–5.

SCHUMACHER, M. 2002. Security Patterns and Security Standards. In *Proceedings of the 7th European Conference on Pattern Languages of Programs*.

SCHUMACHER, M. 2003. *Security Engineering with Patterns*. Springer, Heidelberg, Germany.

SCHUMACHER, M., FERNANDEZ, E. B., HYBERTSON, D., BUSCHMANN, F., AND SOMMERLAD, P. 2005. *Security Patterns - Integrating Security and Systems Engineering*. John Wiley & Sons.

SCOTT, J. AND KAZMAN, R. 2009. Realizing and Refining Architectural Tactics : Availability. Tech. Rep. August, Carnegie Mellon Software Engineering Institute.

U.S. NATIONAL INFORMATION ASSURANCE PARTNERSHIP. 2007. U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness.

WU, D. 2007a. Security functional requirements analysis for developing secure software. Ph.D. thesis, University of Southern California.

WU, W. 2003. Safety Tactics for Software Architecture Design. M.S. thesis, The University of York.

WU, W. 2007b. Architectural reasoning for safety- critical software applications. Ph.D. thesis, University of York.

WYETH, A. M. 2009. Formal Specification of Software Architecture Design Tactics for the Security Quality Attribute. M.S. thesis, California State University.

Quantitative Security Estimation based on Safety Architecture Design Patterns

Christopher Preschern, Nermin Kajtazovic, Andrea Höller, Christian Kreiner

***Abstract*— The increasing connectivity of embedded systems requires more attention to security aspects. Security should not be post-engineered to a system, but should already be considered during system design. However, especially during early design phases it is difficult to judge the impact of high level design decisions, such as the decision for an overall system architecture, on security.**

To provide guidance for the system architecture selection for safety-related systems, we propose a quantitative security assessment method based on the application of design patterns. Based on security threats included in the patters, we calculate a security metric to estimate the patterns' security influence for a specific system. We describe the calculation and application of the proposed security metric with an industrial case study.

***Index Terms*—Design Patterns, Metrics, Safety, Security.**

I. Introduction

Security plays an increasingly important role for embedded systems as they become more and more interconnected. Incidents such as the Stuxnet attack [1] show that even safety-critical systems considered to be secure if not connected to the Internet can become subject to attacks. For safety-critical systems, security often does not play a primary role but is post-engineered after the safety-critical system is designed. An example for this are automation systems which partially have to use standardized insecure protocols. If these systems have to be accessed remotely, a common solution would be to provide a VPN connection and to tunnel the insecure protocol via this connection. However, such systems do not provide any defense in depth and if the VPN security measure can be broken or circumvented, an attacker can influence safety-critical functionality of the system. To analyze and counter relevant attacks and to minimize the effect of successful attacks, security should be considered during system design already. However, especially during early system design, such as the selection of an appropriate architecture meeting the system's safety demands, it is difficult to estimate the effect of these high level design decisions on the resulting system security.

To provide security guidance for the architecture selection for safety-critical systems, we propose a metric to compare the security of safety architecture design patterns. The metric is based on the relevant set of security threats for a pattern and on their probability of occurrence. In this paper we describe with a case study how to calculate and use the security metric in order to choose a system architecture based on design patterns which satisfies both, safety and security needs.

This paper is structured as follows. Section 2 covers related work on security metrics with focus on security metrics for design patterns. Section 3 provides basics on Goal Structuring Notation and safety architecture design patterns. Both will be used in Section 4 where we propose a security metric and apply it to a case study to select a system architecture from a set of safety architecture patterns. Section 5 concludes this work.

II. Related Work

This section presents related work on general state of the art security metrics and in particular on security metrics for design patterns.

A. Security Metrics

Security metrics qualitatively or quantitatively describe the level of security for a system. Literature provides over 900 different security metrics [2]. Some of them are well-established metrics even used in industry and some of them are rather experimental and just described and applied in scientific papers. Several papers give an overview of existing security metrics and try to categorize them. For example, in [3] the authors list existing metrics and divide them into metrics which address a set of security targets or activities, metrics which relate to vulnerabilities, and metrics related to (risk)management. Alternatively security metrics can be categorized according to the main security attributes (e.g. confidentiality) which they address [4]. Another form of overview of existing security metrics which determines attributes such as correctness or measureability to qualify good security metrics is presented in [5]. The authors of [6] suggest a security metric classification based on the software development phase measured by the metric. They obtain the following classes of security metrics (for which we here show representative examples based on [6]):

- *Requirements Phase:* Metrics describing the percentage of fulfilled security requirements or the number of security requirements or misuse cases [7][8]
- *Design Phase:* Metrics describing the number of design decisions or attack surface related metrics [9][10]
- *Implementation Phase:* Metrics describing the number of software failures and in particular security related failures or metrics describing the number of implemented security exception [11]
- *Testing Phase:* Ratio or number of security test cases, reaction of the software to attack patterns [8]
- *Maintenance Phase:* Ratio of software changes due to security incidents or mean time between security incidents [7]
- *System-Level Metrics:* Some security metrics do not just focus on one software development phase, but rather include more than one or even all of them. An

example for such as metric is the Common Criteria security certification. The certification assigns security levels to describe the achieved security of a product which can already be seen as a metric. To provide more detailed metrics based on the standard, the authors of [12] use specific Common Criteria requirements and analyze their achievement in a specific product to describe the product's level of security.

B. Pattern Security Metrics

The authors of [13] evaluate the security of an architecture by considering different misuse patterns. They propose to analyze how many misuse patterns for an architecture can be countered when adding security patterns to improve the architecture. The calculated value then represents the level of security for the applied security patterns. This idea is taken one step further in [14], where the patterns present in an architecture are first automatically mined and then evaluated regarding their influence on security.

The authors of [15] present a similar approach. They map security patterns to quality attributes (confidentiality, availability, ...) which these patterns achieve or improve and estimate the influence of the patterns on these quality attributes. During system design, threat trees are constructed and the risk of the threats is estimated. Patterns which can reduce these risks are then selected. In another paper [16] the authors focus on determining the risk reduction metrics and other security metrics for the patterns.

Halklidis et al. provide the related work which is closest to our proposed security metrics for safety patterns. They analyze a set of security patterns and assign values representing their suitability to counter STRIDE threats [17][18]. These patterns are then taken into consideration when designing a secure architecture. First, they determine possible attacks for a system and structure them in fault trees. Next, they estimate probabilities for the attacks and decide for a set of security patterns by considering how much the attack probabilities can be reduced by the different patterns [19]. Compared to our work, they use patterns from a different domain (not safety) and they do not implicitly cover the threat impact in the pattern structure as it is the case in our work.

III. Basics

This section briefly explains the basics of Goal Structuring Notation diagrams and safety architecture patterns which will both be needed in the following sections.

A. Goal Structuring Notation

Goal Structuring Notation (GSN) is a standardized notation to structurally present elements of an argument (claims/goals, context, evidence, ...) and their relations [20]. With GSN it is possible to link a high level goal to very specific and detailed information/evidence which supports the high level claim/goal.

Figure 1 shows an explanatory example for a GSN diagram. In order to show that the rather general top-level goal ("*The system is safe in case of hardware faults*") is achieved, it is broken down into more specific sub-goals and linked to these sub-goals and their evidence of achievement with arrows. First, for the specific example, an architectural measure (GSN strategy element: "*Redundant Hardware & Output Voting*") is implemented to achieve the top-level goal. Next, sub-goals relevant to achieve the architectural measure come up (e.g. "*Voter works properly*"). If these sub-goals are specific enough, they can be linked to GSN evidence elements (e.g "*Voter test documentation*"). Otherwise they could be further split up. If all sub-goals are related to evidence elements, one gets a complete argument for the achievement of the top-level goal. Further information and examples about GSN can be found in [20].

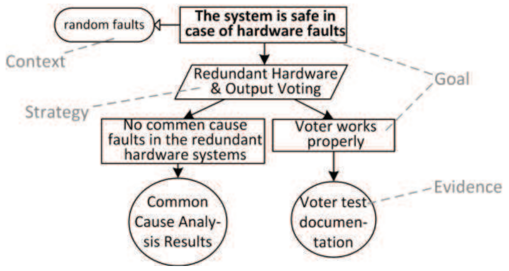

random faults
The system is safe in case of hardware faults
Context
Redundant Hardware & Output Voting
Goal
Strategy
No commen cause faults in the redundant hardware systems
Voter works properly
Evidence
Common Cause Analysis Results
Voter test documentation

Fig. 1. Goal Structuring Notation Example

B. Safety Architecture Patterns

Design patterns describe good solutions for re-occurring problems. There are patterns for software design or architectures in general and also patterns which focus on more specific topics. We will focus on the safety architecture patterns from [21] which provide solutions for high level software or hardware architectures for safety-critical systems.

An example for such an architecture is the Triple Modular Redundancy (TMR) pattern which uses three identical channels and a voter which decides for the majority of the channel outputs. This architecture increases the system's availability and protects from random hardware faults.

The 15 patterns in [21] all contain the following:

- Description in which *Context* they can be applied
- Description which *Problem* the solve
- Description of the *Solution* to that problem including a diagram showing the system's main components
- Description of the *Consequences* when applying the pattern
- *GSN diagram* containing security threats

We will especially focus on the GSN diagram of the patterns which provides a structured presentation of safety-relevant threats. The threats were obtained with the STRIDE threat modeling approach which is described in detail in [22]. The GSN diagrams consist of a collection of goals who's aim is the prevention of attacks related to safety-critical STRIDE threats.

IV. Case Study: Applying the Safety Patterns and Estimating their Effect on Security

In this section we describe an industrial case study for which we consider different safety architectures and compare their security. Figure 2 gives on overview of this section and shows how we will calculate a security metric to guide the architecture selection.

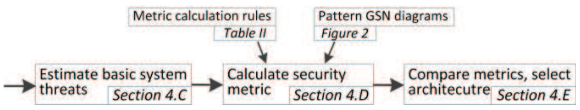


Fig. 2. Process of calculating the security metric

A. System Description

Our case study is an automation system for hydro-power plants. The automation system has to control turbines and has to detect critical states such as too high voltages in the power plant generator. Malfunctions of the hydro-power plant controller could lead to damage of the machinery such as the turbines and could even be a threat to human safety for people operating the power plant. Therefore, the specific system has to be safety-integrity-level 3 (SIL3) certified according to the IEC 61508 safety standard. This standard recommends to use redundant hardware for SIL3 systems. Therefore, we will just consider safety patterns containing redundant hardware as candidates for the system architecture.

Additional hardware constraints for the hydro-power plant controller further narrow down the applicable patterns. For the controller hardware design, at most three independent complex hardware elements (programmable microcontrollers) are available due to cost reasons. Thus, just patterns containing at most three independent hardware channels will be considered.

The above mentioned constraints give us the choice of the following four safety architecture patterns (detailed architecture descriptions can be found in [21]):

– HOMOGENOUS DUPLEX,
  HETEROGENOUS DUPLEX

  The architecture (Figure 3) consists of two homogenous (identical) or heterogenous (diverse) channels which compute outputs from input data. A switch decides which output is actually taken. Normally, the primary channel is used. However, if the fault detector detects a failure in the primary channel, the switch takes the output of the backup channel.

  For the used hardware, the two channels would be implemented on one microcontroller each. One microcontroller would be used to implement the fault detector. The switch would be realized with additional hardware.

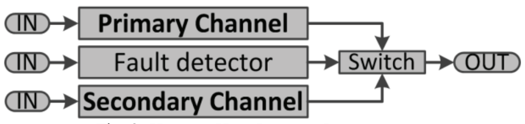


Fig. 3. HOM. or HET. DUPLEX Patterns

– HOMOGENOUS TRIPLE MODULAR REDUNDANCY,
  HETEROGENOUS TRIPLE MODULAR REDUNDANCY

  The architecture consists of three homogenous (identical) or heterogenous (diverse) channels. All channels compute an output and a voter decides for the majority of the outputs.

  For the used hardware, the three channels would be implemented on one microcontroller each. The voter would be realized with additional hardware.

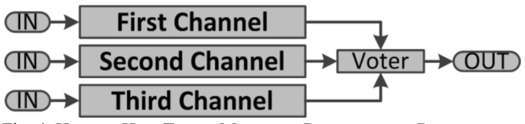


Fig. 4. HOM. or HET. TRIPLE MODULAR REDUNDANCY Patterns

From a safety point of view, all of the four architectures above are applicable for the hydro-power plant controller. Now, we want to evaluate their influence on security to get some more guidance for the architecture selection.

B. Security Threats

The safety architecture patterns of [21] come with a GSN diagram representing safety-relevant threats which were gathered for the patterns with a STRIDE analysis [22]. Figure 5 shows the GSN diagrams for the four considered patterns. We can see that some sets of specific threats (smaller font size in the figure) for the different patterns are similar, but the combination of the threats to an overall security argument (larger font size) is very different. For example, the difference between the HETEROGENOUS and the HOMOGENOUS TMR pattern is that for the heterogenous version, 2 of 3 single channels have to be secure of attacks, while for the homogenous version, no GSN multiplicity element is present in the diagram. This is, because if the channels are identical, an attacker usually can easily attack all three channels as soon as he knows how to compromise one of them. Therefore, the security GSN diagram just considers one channel for the Homogenous TMR pattern.

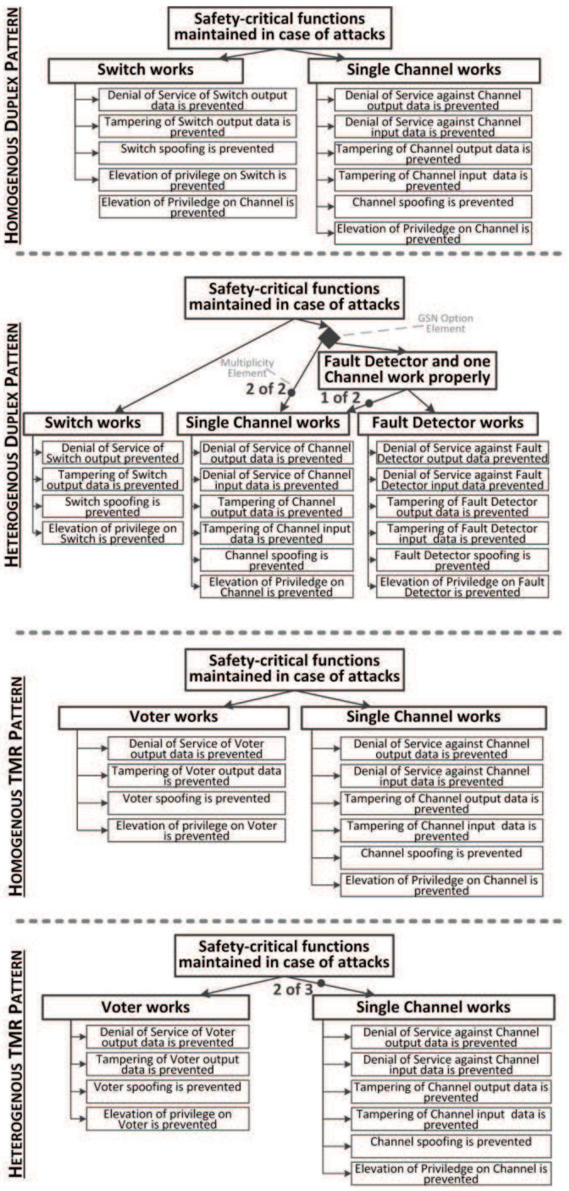

HOMOGENOUS DUPLEX PATTERN
Safety-critical functions maintained in case of attacks
Switch works
Single Channel works
Denial of Service of Switch output data is prevented
Tampering of Switch output data is prevented
Switch spoofing is prevented
Elevation of privilege on Switch is prevented
Elevation of Priviledge on Channel is prevented
Denial of Service against Channel output data is prevented
Denial of Service against Channel input data is prevented
Tampering of Channel output data is prevented
Tampering of Channel input data is prevented
Channel spoofing is prevented
Elevation of Priviledge on Channel is prevented
HETEROGENOUS DUPLEX PATTERN
Safety-critical functions maintained in case of attacks
GSN Option Element
Multiplicity Element
2 of 2
1 of 2
Fault Detector and one Channel work properly
Switch works
Single Channel works
Fault Detector works
Denial of Service of Switch output prevented
Tampering of Switch output data is prevented
Switch spoofing is prevented
Elevation of privilege on Switch is prevented
Denial of Service of Channel output data is prevented
Denial of Service of Channel input data is prevented
Tampering of Channel output data is prevented
Tampering of Channel input data is prevented
Channel spoofing is prevented
Elevation of Priviledge on Channel is prevented
Denial of Service against Fault Detector output data prevented
Denial of Service against Fault Detector input data prevented
Tampering of Fault Detector output data is prevented
Tampering of Fault Detector input data is prevented
Fault Detector spoofing is prevented
Elevation of Priviledge on Fault Detector is prevented
HOMOGENOUS TMR PATTERN
Safety-critical functions maintained in case of attacks
Voter works
Single Channel works
Denial of Service of Voter output data is prevented
Tampering of Voter output data is prevented
Voter spoofing is prevented
Elevation of privilege on Voter is prevented
Denial of Service against Channel output data is prevented
Denial of Service against Channel input data is prevented
Tampering of Channel output data is prevented
Tampering of Channel input data is prevented
Channel spoofing is prevented
Elevation of Priviledge on Channel is prevented
HETEROGENOUS TMR PATTERN
Safety-critical functions maintained in case of attacks
2 of 3
Voter works
Single Channel works
Denial of Service of Voter output data is prevented
Tampering of Voter output data is prevented
Voter spoofing is prevented
Elevation of privilege on Voter is prevented
Denial of Service against Channel output data is prevented
Denial of Service against Channel input data is prevented
Tampering of Channel output data is prevented
Tampering of Channel input data is prevented
Channel spoofing is prevented
Elevation of Priviledge on Channel is prevented

Fig. 5. GSN diagrams for safety architecture patterns (based on [21])

C. *Security Risk Estimation*

We have seen in Figure 3 that many of the threats are similar for the different patterns. We now rate these general threats without yet knowing which specific architecture will be chosen. We conducted a meeting with system architects and security experts of our industrial partner to estimate the probability of attacks related to specific threats for the hydro-power plant controller. Note that we do not estimate the attack impact (which is common in security risk estimation like [22]), because the impact is already implicitly covered in the position of the threat in the GSN diagram. This is an advantage of the presented approach compared to pattern security metrics described in literature. The results of the probability of attack estimation made by the industrial partner is shown in Table I where the introduced "Goal Confidence" (probability that the threat-related GSN goal is achieved) is the converse probability to the estimated "Probability of Attack" and the values (none, low, medium, high) are substituted with (0.00, 0.01, 0.05,0.20). These values were chose based similar estimations in literature [15][19].

TABLE I: GENERAL THREATS FOR ALL CONSIDERED ARCHITECTURES

Threat	**Prob.of Attack**	**Goal-Conf.**	**Comments**
Denial of Service, microcontroller output data	none	1.00	The outputs are always hardwired to the actors, switches or voters
Denial of Service of microcontroller input data	low	0.99	The inputs are received via a separate LAN
Tampering of microcontroller output data	none	1.00	The outputs are always hardwired to the actors, switches or voters
Tampering of microcontroller input data	low	0.99	The inputs are received via a separate LAN
Elevation of privilege on microcontroller	high	0.80	Controller connected to the local LAN; OS could be compromised
Microcontroller spoofing	medium	0.95	Users often just use weak passwords
Denial of Service of switch/ voter output data	none	1.00	The outputs are hardwired
Tampering of switch/voter output data	none	1.00	The outputs are hardwired
Elevation of privilege on switch/voter	low	0.99	Unlikely, because it is a simple hardware element
Switch/Voter spoofing	none	1.00	The outputs are hardwired

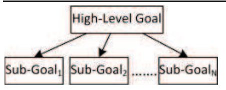
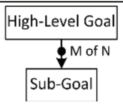
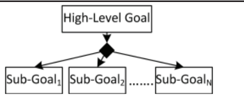
D. *Goal Confidence*

With the "Goal Confidence" for the threat goals estimated in Table I, we can now compute an overall metric for the security of the pattern architectures. We simply use probability theory to propagate the goal confidence to higher level goals. Table II shows the calculation rules.

With the established rules we can now calculate the goal confidence of the top-level goals for the security GSNs of the four considered safety architecture patterns. First, we calculate the goal confidence for "*Single Channel works properly*" or "*Fault Detector works properly*" (goals from Figure 5):

$$GC_{ch/fd} = \prod_{i=1}^{6} GC_{microcontroller-threats}$$
$$= GC_{\mu C-out-dos} * GC_{\mu C-in-dos} * GC_{\mu C-out-tampering}$$
$$* GC_{\mu C-in-tampering} * GC_{\mu C-eop} * GC_{\mu C-spoofing}$$
$$= 1.00 * 0.99 * 1.00 * 0.99 * 0.80 * 0.95 = \mathbf{74.5\%}$$

TABLE II: GOAL CONFIDENCE FORMULAS

GSN Relation	Goal-Confidence
 N subgoals have to be met to meet the high-level goal	$GC = \prod_{i=1}^{N} GC_{sub\_i}$
 M out of N subgoals have to be met to meet the high-level goal	$GC = \sum_{k=M}^{N}\left(\binom{N}{k}(GC_{sub})^{k}(1-GC_{sub})^{N-k}\right)$ (formula taken from [23])
 At least one of N subgoals has to be met to meet the high-level goal	$GC = 1 - \prod_{i=1}^{N}(1 - GC_{sub\_i})$

Next we calculate the goal confidence for "*Switch works properly*" or "*Voter works properly*":

$$GC_{s/v} = \prod_{i=1}^{4} GC_{switch/voter-threats}$$
$$= GC_{sv-dos} * GC_{sv-tampering} * GC_{sv-eop} * GC_{sv-spoofing}$$
$$* GC_{\mu C-in-tampering} * GC_{\mu C-eop} * GC_{\mu C-spoofing}$$
$$= 1.00 * 1.00 * 0.99 * 1.00 = \mathbf{99.9\%}$$

Next we calculate the goal confidence for the top-level goals for our four considered architectures. We simply apply the equations from Table II according to the GSN diagrams from Figure 3 to obtain the goal confidence values for the HOMOGENOUS TMR, HETEROGENOUS TMR, and the HOMOGENOUS DUPLEX system.

$$GC_{\text{hom}-tmr} = GC_{s/v} * GC_{ch/fd}$$
$$= 0.99 * 0.74 = \mathbf{73.7\%}$$

$$GC_{het-tmr} = GC_{s/v} \sum_{k=2}^{3}\binom{3}{k}(GC_{ch/fd})^{2}(1-GC_{ch/fd})^{3-k}$$
$$= GC_{s/v} * \left[3(GC_{ch/fd})^{2} - 2(GC_{ch/fd})^{3}\right]$$
$$= 0.99 * \left[3 * 0.74^{2} - 2 * 0.74^{3}\right] = \mathbf{83.0\%}$$

$$GC_{\text{hom}-duplex} = GC_{s/v} GC_{ch/fd}$$
$$= 0.99 * 0{,}74 = \mathbf{73.7\%}$$

For the HETEROGENOUS DUPLEX system we first calculate the goal confidence of the "*Fault Detector and one Channel work properly*" goal and then the overall goal confidence.

$$GC_{fd-and-ch} = GC_{ch/fd}\left[\sum_{k=1}^{2}\binom{2}{k}(GC_{ch/fd})^{k}(1-GC_{ch7fd})^{2-k}\right]$$
$$= GC_{ch/fd}\left[2 * GC_{ch/fd} - (GC_{ch/fd})^{2}\right]$$
$$= 0.74 * \left[2 * 0.74 - 0.74^{2}\right] = \mathbf{69.7\%}$$

$$GC_{het-duplex} = GC_{s/v}\left[1 - \prod_{i=1}^{2}(1 - GC_{sub\_i})\right]$$
$$= GC_{s/v}\left[1 - (1 - GC_{ch/fd}{}^{2} *)(1 - GC_{fd-and-ch})\right]$$
$$= 0.99 * \left[1 - (1 - 0.74^{2})(1 - 0.69)\right] = \mathbf{85.7\%}$$

E. Architecture Decision

Based on the probability estimation for threat-related attacks, we now have the following security metrics for our four considered architectures:

- Homogenous TMR: 73.7%
- Heterogenous TMR: 83.0%
- Homogenous Duplex: 73.7%
- Heterogenous Duplex: 85.7%

These values represent the probability of an attacker tampering with the safety functionality of the system. However, one has to be careful when interpreting these values. The absolute values are highly dependent on our mapping of the (none, low, medium, high) terms used during the threat probability estimation to quantitative values. This means that the absolute values are not well suited to directly represent the exact probabilities for attacks. However, when comparing the values for the different architectures, the calculated security metrics can be very useful. For example, we can see that the Homogenous TMR pattern for our case study is less secure than the Heterogenous TMR pattern. This is not very surprising, because obviously it is more difficult for an attacker to attack three diverse systems compared to three identical systems. Still, for other architecture pairs, the security metrics do give valuable information. For example, the metrics say that the Heterogenous TMR pattern is less secure then the Heterogenous Duplex pattern in the case of our hydro-power plant controller with its specific threat probability estimation. This information is not obvious and can give guidance for preferring one safety architecture over another. It is also interesting to compare the security metrics of the patterns to the security metric of a single microcontroller (which would be the architecture for the system without applying any of the patterns):

- Homogenous TMR: $\frac{GC_{\text{hom}-TMR}}{GC_{ch/fd}} = \frac{73.7\%}{74.5\%} = 0.99$
- Heterogenous TMR: $\frac{GC_{het-TMR}}{GC_{ch/fd}} = \frac{83.0\%}{74.5\%} = 1.11$
- Homogenous Duplex: $\frac{GC_{\text{hom}-duplex}}{GC_{ch/fd}} = \frac{73.7\%}{74.5\%} = 0.99$
- Heterogenous Duplex: $\frac{GC_{het-duplex}}{GC_{ch/fd}} = \frac{85.7\%}{74.5\%} = 1.15$

These values represent the relative security improvement when applying the pattern. We can see that the homogenous version of the patterns actually decrease the overall security of the system. This is not surprising, because they increase the attack surface for the system. The heterogenous versions also increase the attack surface, but they also provide more security, because compared to a single microcontroller, they provide diverse systems which requires an attacker to break more than one of the microcontrollers.

For the hydro-power plant controller, the information about the influence of the architecture on security supported the decision to implement the Heterogenous Duplex pattern. Figure 6 shows the high-level architecture of the resulting hydro-power plant controller.

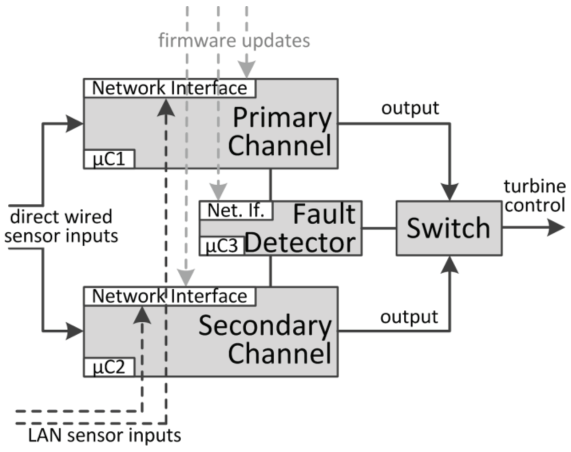


Fig. 6. Implemented Heterogenous Duplex architecture for the hydro-power plant controller

F. Discussion

We calculated security metrics for four patterns; however, with the calculation rules given in this paper, it is no problem to calculate the security metrics also for the other patterns in [21]. This would be necessary, for example, if there were different hardware constraints for a system architecture. Our choice for one of the four considered architectures of course not just effects system security, but even has a bigger effect on other quality attributes such as safety, availability, or system costs which might be more important drivers for the design decision. Still, the security metric allows to already consider security at early stage architecture design and provides the possibility to quantitatively compare the security of different architectures. In particular in some cases, as in our case study, where from the point of view of other quality attributes several different architectures are possible candidates, the security metrics can help making a decision.

A drawback of the proposed security metric is that someone who is familiar with the system has to estimate the probability of attacks related to some basic threats. However, a risk estimation step is usually required for any kind of quantitative security metrics. The advantage of the proposed approach is, that one just has to estimate the probability of attacks and not their impact, because the information about the impact is already present in the safety architecture patterns.

V. Conclusion

In this paper we showed with a case study how security metrics can help to make decisions for early system architecture design. We estimated the probability of general attacks for a case study and calculated a resulting security metric when applying different safety architecture patterns in order to compare them and decide for one of them. Apart from the design guidance by the patterns as good solutions to specific problems, with the described approach a system designer now has the possibility to quantitatively estimate the influence of safety architecture selection on security. The presented method to compare the security of different architectures aids the decision for a specific architecture from a security point of view.

For future work it would be interesting to apply the security metric to other case studies or to apply the metric to GSN diagrams of other domains. The metric could even be extended to evaluate both, safety and security, of an architecture if appropriate safety GSN diagrams for the architecture are available. Another point for future work is to thoroughly evaluate the proposed security (for example with a sensitivity analysis) metric. At this point it is difficult to compare the metric to security metrics in literature, because the metrics in literature do not describe the security of general architectures and require more detailed security assumptions/information about the system (such as the impact and probability of detailed attacks). However, in the long term we plan to evaluate the proposed metric by analyzing the number of incidents on different systems which apply the safety architecture patterns and we expect to obtain a relation to our security metric.

With the approach described in this paper we provide a way to easily compare the level of security when applying safety design patterns. With this method we aim to increase the awareness and consideration of security problems in the safety domain.

References

[1] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon", *IEEE Security and Privacy*, vol. 9, issue 3, pp. 49-51, 2011
[2] D. S. Herrmann, *Complete Guide to Security and Privacy Metrics*, 1st ed. Auerbach Pub, 2007
[3] J. Bayuk, A. Mostashari, "Measuring systems security", Systems Engineering, vol. 16, issue 1, pp. 1-14, 2013
[4] D. Mellado, E. Fernandez-Medina, M. Piattini, "A comparison of software design security metrics", *Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA10*, ACM Press, 2010
[5] R. M. Savola, "Quality of security metrics and measurements", *IEEE Computers & Security*, volume 37, pp. 78-90, 2013
[6] S. Jain., M. Ingle.: "A review of security metrics in software development process", International Journal of Computer Science and Information Technologies, 2011
[7] K. Sultan, A. En-Nouaary, A.H.L.: "Catalog of Metrics for Assessing Security Risks of Software throughout the Software Development Life Cycle", *Proceedings of the International Conference on Information Security and Assurance*, IEEE, 2008
[8] J. Allen, "Measuring software security". Technical report, Software Engineering Institute, Carnegie Mellon University, 2009
[9] J. Stuckman, J. Purtilo.: "Comparing and Applying Attack Surface Metrics", *Proceedings of the 4th International Workshop on Security Measurements and Metrics*, ACM, 2012
[10] P. K. Manadhata, J. M. Wing, "An attack surface metric", *IEEE Transactions on Software Engineering*, volume 37, pp. 371–386, 2011
[11] I. Chowdhury, B. Chan, M. Zulkernine, "Security Metrics for Source Code Structures", *Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems*, ACM, 2008
[12] A. Hunstad, J. Hallberg, R. Andersson, "Measuring IT security - a method based on common criteria's security functional requirements", *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop*, IEEE, 2004
[13] E. B. Fernandez, N. Yoshioka, H. Washizaki, M. VanHilst.: "Measuring the Level of Security Introduced by Security Patterns", *Proceedings of the International Conference on Availability, Reliability and Security*, IEEE, 2010
[14] A. M. Krishna, N. S. Goud, A. K. Prasad, "Software Security Architectures Architecture Mining", *International Journal of Engineering Associates*, volume 2, issue 3, pp. 32-36, 2013
[15] A. Yautsiukhin, R. Scandariato: "Towards a quantitative assessment of security in software architectures", *Proceedings of the 13th Nordic-Workshop on Secure IT Systems (NordSec)*, 2008
[16] T. Heyman, R. Scandariato, C. Huygens, W. Joosen.: "Using Security Patterns to Combine Security Metrics", *Proceedings of the Third International Conference on Availability, Reliability and Security*, IEEE, 2008
[17] S. Halkidis, A. Chatzigeorgiou, G. Stephanides: "A qualitative evaluation of security patterns", *Proceedings of the 6th International Conference on Information and Communications Security*, Springer, 2004
[18] S. Halkidis, A. Chatzigeorgiou, G. Stephanides: "A qualitative analysis of software security patterns", Computers & Security, volume 25, issue 5, pp. 379–392, 2006
[19] S. Halkidis., N. Tsantalis., A. Chatzigeorgiou., G. Stephanides.: "Architectural Risk Analysis of Software Systems Based on Security Patterns", *IEEE Transactions on Dependable and Secure Computing*, volume 5, issue 3, pp. 129-142, 2008
[20] J. Spriggs, *GSN - The Goal Structuring Notation: A Structured Approach to Presenting Arguments*, 1st edition, Springer, 2012
[21] C. Preschern, N. Kajtazovic, C. Kreiner, "Security Analysis of Safety Patterns" Proceedings of the 20th Conference on Pattern Languages of Programs (PLoP), ACM Press, 2013
[22] M. Howard, D. LeBlanc, *Writing Secure Code*. 1st edition, Microsoft Press, 2003
[23] W. Kuo, M. Zuo, *Optimal Reliability Modeling: Principles and Applications*. 1st edition, John Wiley & Sons, 2003

Christopher Preschern received the Master's degree in telematics from Graz University of Technology in 2011, focusing on software product lines, automation systems and IT-security. He is working toward the Ph.D. degree in electrical engineering at the Institute for Technical Informatics, Graz University of Technology. His research focuses on design patterns and their relationship to non-functional quality attributes, in particular safety and security.

Nermin Kajtazovic received his Master's degree in telematics form Graz University of Technology in 2011. He focused on variability management in component-based architectures for the automotive domain. He is a PhD student at the Institute for Technical Informatics, Graz University of Technology. His research is related to component-based systems in the safety domain.

Andrea Höller received her Master's degree in telematics form Graz University of Technology in 2013. She focused on implementation and attacks elliptic curve cryptography hardware. Currently, she is a PhD student at the Institute for Technical Informatics, Graz University of Technology. Her research is related to multi-core technology, safety-critical systems, and diversity measures.

Christian Kreiner graduated and received the Ph.D. degree in electrical engineering from Graz University of Technology, in 1991 and 1999, respectively. From 1999 to 2007, he served as head of the R&D Department, Salomon Automation, Austria, focusing on software architecture, technologies, and processes for logistics software systems. He was in charge to establish a company-wide software product line development process and headed the product development team. There, he led and coordinated a long-term research program together with the Institute for Technical Informatics of Graz University of Technology. There, he currently leads the Industrial Informatics and Model-based Architectures group. His research interests include systems and software engineering, software technology, and process improvement.

2012 38th Euromicro Conference on Software Engineering and Advanced Applications

Structuring Modular Safety Software Certification by Using Common Criteria Concepts

Christopher Preschern
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
Email: christopher.preschern@tugraz.at

Kurt Dietrich
Institute for Applied Information Processing and Communications
Graz University of Technology
Graz, Austria
Email: kurt.dietrich@iaik.tugraz.at

***Abstract*—Safety and security certification are time and money consuming tasks. Changes to certified systems usually require re-certification of the whole product. Modular certification approaches applied to the safety and security domain aim at reducing these costs.**

In this paper, modular certification concepts with focus on IEC 61508 safety certification are analyzed and an approach for structuring the modular certification process by providing detailed requirements is suggested. We gather requirements from the security domain in order to fulfill objectives which have to be reached to enable modular safety certification. Functional requirements are taken from the Common Criteria Separation Kernel Protection Profile and assurance requirements are taken from a Common Criteria class responsible for compositional security certification.

***Keywords*-safety; security; modular certification; IEC 61508; Common Criteria; Separation Kernel Protection Profile**

I. Introduction

Safety and security critical devices require certification according to standards, such as IEC 61508 for safety and Common Criteria for security. Both of these certifications require rigorous development and maintenance processes why nowadays, certification takes a big part of product development costs. For reducing these costs it is very desirable to find more efficient approaches to achieve certification.

Because safety and security have several common aims, like system integrity for example [1], certifications for safety and security also have some processes in common. Several steps for safety and security certification could be combined or artifacts from one certification could be used for the other to reduce overall certification costs for systems which have to be certified for both, safety and security [2]. There are suggestions in literature on how to merge safety and security certification processes, although current certification standards do not explicitly allow that. There even have been attempts to provide a basis for combined safety and security certification in form of a suggestion for a safety and security standard [3]. Even if a system does not need to be certified for both safety and security, there still is a lot of potential that safety and security certification processes could benefit from one another by using well established methods from the other certification process.

An example where safety certification could benefit from security standards is modular certification, where system components can be certified independently from one another in order to keep certification costs low. This allows effective certification and in particular effective re-certification of modified systems. The IEC 61508 safety certification standard allows modular certification, but it is not very well described in the standard which requirements have to be met in order to be allowed to certify components independently from one another. The standard lacks of a structured approach for system developers to show the safety certification authority whether modular certification can be used for certifying a system.

In this paper, we suggest an approach providing a structured basis for showing an IEC 61508 certification authority whether modular safety certification can be applied. The current modular safety certification process requires the developer to show time and space partitioning between components in order to certify them independently. The standard does not cover how this can be achieved in detail. We suggest to use Common Criteria security certification requirements taken from a Protection Profiles with similar aims, as a basis to show software component independence for modular safety certification. Protection Profiles are collections of security certification requirements for specific product types, such as the product type of separation kernels for example.

Artifacts from the Separation Kernel Protection Profile (SKPP) can build a basis for time and space partitioning requirements in the safety domain. The SKPP includes a detailed collection of requirements necessary for time and space partitioning and has been acknowledged by a big community; therefore, it is more likely to be complete than solutions that are developed individually.

II. Modular Certification

This section shows state of the art regarding modular safety and security certification with focus on the IEC 61508 safety standard.

Rushby [4] presents the idea of modular certification and shows its basic requirements. To provide modular certification, isolation between the modules has to be enforced so that safety and security attributes can still be guaranteed when modules are being composed. To achieve that, microkernels are commonly used to separate applications. Rushby [5] draws connections between the safety partitioning property and security efforts by relating security attributes such as process security, data flow security, access control and denial of service to the safety partitioning process. The connections indicate that the problems in the two domains are rather similar and that probably a common solution for modular certification can be used for both, the safety and the security domain.

A. Modular Security Certification

Evolving from Rushby's requirements, nowadays the Multiple Independent Levels of Security (MILS) architecture is commonly used for separating security modules to allow modular certification. MILS is based on different layers of trust, where upper layers are based on lower layers, but do not depend on anything else in the system. In the security domain such systems are implemented as separation kernels which ensure that software running on top of the kernel is partitioned into independent components which cannot influence each other [6]. Such systems can be certified to the Common Criteria standard, which offers an approved Protection Profile containing requirements for separation kernels. Modular certification for secure systems is possible since version 3.0 of Common Criteria which contains an assurance class addressing compositional certification (the ACO:Composition class) [2] [7].

B. Modular Safety Certification

Regarding safety, modular certification is often applied to the avionics domain where the well established Integrated Modular Avionics architecture allows to build and certify components separately from each other [8] [9]. The architecture is based on a separation kernel which handles data and resource partitioning and protects tasks from unintended interaction.

Most approaches for modular safety certification are based on a certified operating system which monitors resource allocation and task partitioning [5]. McDermid et. al. [10] suggest an approach called LISA (Low-level Interaction Safety Analysis) to analyze partitioning capabilities of operating systems for safety-critical applications. The approach analyzes system resources and possible failures with special focus on the effect of hardware failures on software components.

Recently also other approaches, which are not based on the underlying operating system, are used to achieve modular safety certification. Amey et. al. [11] describes a modular certification concept, where the SPARK programming language is used in combination with static code analysis to provide a proof that one part of the code cannot influence other parts. Also [12] shows how the usage of a safe programming language subset in combination with static code analysis enables robust partitioning. Delange et. al. [13] present an approach where the system architecture is modeled with an architecture description language and architectural constraints regarding safety partitioning can be checked.

C. Modular Safety Certification According to IEC 61508

The IEC 61508 standard [14] allows modular certification and covers several architectural aspects allowing later software modifications. For modifications, the effect on other system components has to be analyzed. For hardware, the Failure Mode and Effects Analysis (FMEA) is used to asses a system regarding safety and to check which parts of a system can influence one another. For software, several effect analysis techniques have been proposed in literature [15] and a very basic explanation for a software effect analysis is given in IEC 61508-3 appendix F. Depending on the result of this effect analysis, just parts of the system or even the whole system have to be re-certified.

IEC 61508-3 appendix F describes objectives (time and space partitioning) which have to be achieved for software components in order to allow modular certification. These objectives include that data cannot be modified by other modules and that a module cannot block resources (e.g. processing time) in a way that the safety functionality of another module fails.

The IEC 61508 standard describes no detailed requirements for the partitioning mechanism. The only information given, is that time and space partitioning has to be ensured. By not giving clear requirements, the standard leaves the task of gathering these requirements up to the system developer. We suggest to use requirements already collected and approved in the security domain, because a security certified separation kernel must meet similar objectives.

III. Developing a Structured Method to Achieve Modular Safety Certification

In this section we propose a well structured approach to show software component independence which allows modular safety certification. We gather security requirements and present a method how to use these requirements to make a decision on which parts of a system modular certification can be applied.

The SKPP [16] describes requirements for separation kernels which are used in the security domain to achieve modular certification. Some of the functional separation kernel requirements can be used as requirements to prove independence between software safety components. We analyzed the SKPP to filter out time and space partitioning

Table II
DESCRIPTION OF ALL PARTS OF THE COMMON CRITERIA ACO:COMPONENTS CLASS [7]

Name	Description
ACO_ COR	This family addresses the requirement to demonstrate that the base component can provide an appropriate level of assurance for use in composition.
ACO_ DEV	This family sets out requirements for a specification of the base component in increasing levels of detail. Such information is required to gain confidence that the appropriate security functionality is provided to support the requirements of the dependent component (as identified in the reliance information).
ACO_ REL	The purpose of this family is to provide evidence that describes the reliance that a dependent component has upon the base component. This information is useful to persons responsible for integrating the component with other evaluated IT components to form the composed TOE, and for providing insight into the security properties of the resulting composition. This provides a description of the interface between the dependent and base components of the composed TOE that may not have been analyzed during evaluation of the individual components, as the interfaces were not TSF interfaces of the individual component TOEs.
ACO_ CTT	This family requires that testing of composed TOE and testing of the base component, as used in the composed TOE, is performed.
ACO_ VUL	This family calls for an analysis of vulnerability information available in the public domain and of vulnerabilities that may be introduced as a result of the composition.

requirements which are relevant for the safety effect analysis. The Protection Profile gathers these requirements in two objectives (see Table I) which address resource allocation and isolation between tasks.

The ACO:Components class of Common Criteria Part 3 describes assurance requirements for composed systems. A basic description of its content is shown in Table II. The ACO_COR requirement can also be found in the IEC 61508 safety standard in relation to time and space partitioning mechanisms, which have to be certified to a level at least as high as the component using its service. The ACO_DEV and ACO_REL requirements describe the interfaces and interactions between components, whis is not described in such detail in the safety standard. The ACO_CTT requirements are already covered in the safety domain by integration testing and ACO_VUL addresses a vulnerability analysis which is unnecessary for safety systems. From the ACO:Components class, ACO_COR, ACO_DEV, and ACO_REL provide useful information for the software component effect analysis.

The functional requirements taken from the SKPP and the ACO_COR requirement can serve as a basis to show time and space partitioning. A time and space partitioned system need not be re-certified as a whole if components are modified, because the partitioning mechanism assures that other components are not affected. Components directly affected by the change in the system still have to be re-verified in a time and space partitioned architecture, because

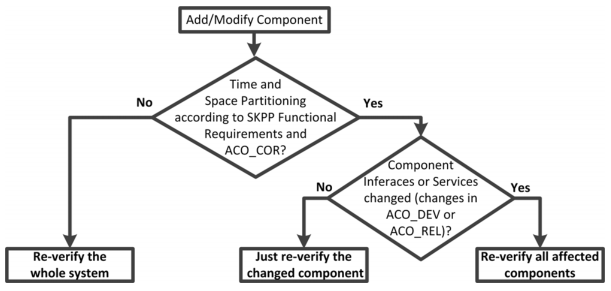

Add/Modify Component
Time and Space Partitioning according to SKPP Functional Requirements and ACO_COR?
No
Yes
Component Inferaces or Services changed (changes in ACO_DEV or ACO_REL)?
No
Yes
Re-verify the whole system
Just re-verify the changed component
Re-verify all affected components

Figure 1. Effect of component change on re-verification effort

their functionality depends on the modified component. If the component communication is based on well defined interfaces and the components guarantee services at these interfaces, then such components would not have to be re-verified if the interface communication and service guarantee stays the same. These interface criteria are defined by the ACO_DEV and ACO_REL requirements from the ACO:Components class.

For systems which meet the time and space partitioning requirements in Table I and the ACO_COR requirement, not all parts have to re-certified after modifications. Just the modified components and the components depending on services of modified or affected components have to be re-verified. If, additionally, the requirements ACO_DEV and ACO_REL are not changed for the modified component, just the modified component itself has to be re-verified and any other parts of the system need no re-verification, because the service provided by the component does not change and, therefore, any other components cannot be affected. Figure 1 shows a decision tree presenting the outcome of the software affect analysis for modular safety certification by using the requirements form the SKPP and the ACO:Components class.

IV. CONCLUSION

In this paper we showed how modular certification in the safety domain can be achieved by using methods from the security domain. Common Criteria compositional assurance requirements and SKPP partitioning requirements build a basis for a structured approach to analyze dependencies for software components of a safety system. Our approach can be taken as a reference for safety system developers to show a safety certification authority that a system meets the necessary requirements for modular certification. Compared to the software effect analysis of IEC 61508, our approach is more detailed and provides a more structured way to argument for component independence. We do not propose an alternative way for modular certification, compared to the IEC 61508 standard we propose a refined method.

We believe that with this method the safety certification effort, in particular the safety re-certification effort, can be

Table I
SKPP OBJECTIVES RELATED TO TIME AND SPACE PARTITIONING [16]

Objective / ***Requirement***	**Description**
O.RESOURCE_ ALLOCATION / *FRU_RSA.2*	Allocation limits are enforced for the minimum and maximum amount of memory and processing time available to a partition. Allocation requirements for system memory are based on the minimum and maximum simultaneous memory usage by each individual partition at any given time. Allocation limits on processing time are based on the minimum and maximum CPU usage by each individual partition over a specific time interval. A refinement to the wording of the choices provided by the CC in the selection operation was made. The allocation is made to the partition, which is inclusive of subjects and exported resources (there are no 'users' in the context of the separation kernel allocation of resources).
O.SUBJECT_ ISOLATION / *FDP_IFC.2* *FDP_IFF.1* *FPT_SEP.3*	This objective requires the Target Of Evaluation (TOE) to establish security domains for subjects where each subject is completely isolated from every other subject. This complete isolation is the default configuration that is established by the TOE Security Functionality (TSF). Where flows between subjects are specified by the configuration data and mediated by the TSF throughout the execution session, the scope of this objective expands and must also ensure that no unauthorized information flows can occur, which may result in one subject interfering with another. FDP_IFC.2 and FDP_IFF.1 combine to define the scope of the partitioned information flow policy to be enforced by the TSF, and the rules implemented by the TSF to enforce the policy. This enforcement capability of the TSF ensures that strict isolation of a subject is preserved where no flows to/from the subject are allowed, and ensures that only authorized information flows as specified by the configuration data are allowed. FPT_SEP.3 satisfies this objective by requiring the TSF to enforce separation between the security domains of all subjects in the TSF scope of control, thus ensuring that subjects cannot access or manipulate other subject's services and resources in violation of the TOE security policy. The security domain of a subject includes the services and exported resources that the particular subject is allowed to use.

reduced and time and costs can be saved. The possibility of modular certification motivates more intensive software maintenance, because it requires less costs to re-certify product modifications. This is highly desirable, because in the safety domain, software maintenance becomes more important due to the growing complexity of safety critical systems. Modular certification also provides a basis for modular software concepts such as component based systems which are, nowadays, not yet commonly used in the safety domain, but are already handled in literature and appear to be a promising field of research.

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[2] C. Taylor, "Component Based Common Criteria Certification," in *Systems, Software and Technology Conference*, 2006.

[3] B. Dobbing and S. Lautieri, "SafSec : Integration of Safety & Security Certification," Praxis High Integrity Systems, Tech. Rep., 2006. [Online]. Available: http://www.altran-praxis.com

[4] J. M. Rushby, "Design and verification of secure systems," *ACM SIGOPS Operating Systems Review*, vol. 15, no. 5, pp. 12–21, Dec. 1981.

[5] J. Rushby, "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance," NASA Langley, Tech. Rep., 2000. [Online]. Available: http://www.tc.faa.gov/its/worldpac/techrpt/ar99-58.pdf

[6] J. Alves-Foss, C. Taylor, and P. Oman, "A Multi-layered Approach to Security in High Assurance Systems," in *Proceedings of the 37th Hawaii International Conference on System Sciences*. IEEE, 2004.

[7] Common Criteria, "Common Criteria Standard v3.1," http://www.commoncriteriaportal.org/cc/, 2009.

[8] G. Romanski, "Safe and Secure Partitioned Systems and Their Certification," in *30 IFAC Workshop on Real-Time Programming*, vol. 30, 2009, pp. 12–14.

[9] J. Windsor and K. Hjortnaes, "Time and Space Partitioning in Spacecraft Avionics," in *2009 Third IEEE International Conference on Space Mission Challenges for Information Technology*. IEEE, Jul. 2009, pp. 13–20.

[10] J. A. McDermid and D. J. Pumfrey, "Assessing the Safety of Integrity Level Partitioning in Software," in *Proceedings of the Eighth Safety-critical Systems Symposium*. Springer, 2000, pp. 134–152.

[11] P. Amey, R. Chapman, and N. White, "Smart Certification Of Mixed Criticality Systems," in *Proceedings of Ada-Europe'2005*, 2008, pp. pp.144–155.

[12] B. Dobbing, "Building partitioned architectures based on the Ravenscar profile," *ACM SIGAda Ada Letters*, vol. XX, no. 4, pp. 29–31, Dec. 2000.

[13] J. Delange, L. Pautet, and P. Feiler, "Validating safety and security requirements for partitioned architectures," in *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*. Springer, 2009.

[14] International Electrotechnical Commission, "IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems," 1999.

[15] J. McDermid, "Software hazard and safety analysis," in *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer Berlin / Heidelberg, 2002.

[16] U.S. National Information Assurance Partnership, "U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness," 2007. [Online]. Available: http://www.niap-ccevs.org/pp/pp_skpp_hr_v1.03.pdf

Bibliography

[AKA12] Pablo Oliveira Antonino, Thorsten Keuler, and Pablo Antonino. Towards an Approach to Represent Safety Patterns. In *The Seventh International Conference on Software Engineering Advances (ICSEA)*, pages 228–237, 2012.

[Ale79] Christopher Alexander. *The Timeless Way of Building.* Oxford University Press, 1979.

[Arm10] Ashraf Armoush. *Design patterns for safety-critical embedded systems.* PhD thesis, RWTH Aachen University, 2010.

[Bab07] Muhammad Ali Babar. Improving the Reuse of Pattern-Based Knowledge in Software Architecting. In *EuroPLoP*, pages 7–11. Lero, Ireland, 2007.

[BB98] Peter Bishop and Robin Bloomfield. A Methodology for Safety Case Development. In *Proceedings of the Sixth Safety-critical Systems Symposium.* Springer, 1998.

[BBD+12] Pierre Bieber, Jean-Paul Blanquart, Gilles Descargues, Michael Dulucq, Yannick Fourastier, Eric Hazane, Mathias Julien, Laurent Leonardon, and Gabrielle Sarouille. Security and Safety Assurance for Aerospace Embedded Systems. In *Embedded Real Time Software And Systems (ERTSS)*, 2012.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* Addison-Wesley Longman, 2003.

[Ber10] Vincent Bernard. Modsafe deliverable report wp1 d1.2. Technical report, MODSafe FP7 Project, 2010.

[Bit07] Friedemann Bitsch. *Verfahren zur Spezifikation funktionaler Sicherheitsanforderungen fr Automatisierungssysteme in Temporallogik.* PhD thesis, Universität Stuttgart, 2007.

[BMRS96] Frank Buschmann, Regine Meunier, Hans Rohnert, and Peter Sommerlad. *Pattern-Oriented Software Architecture: A System of Patterns.* John Wiley & Sons, 1996.

[BNS13] Robin Bloomfield, Kateryna Netkachova, and Robert Stroud. Security-Informed Safety: If It's Not Secure, It's Not Safe. In *Proceedings of the 5th International Workshop on Software Engineering for Resilient Systems*, 2013.

[Bro08] Benjamin Brogsol. Safety and Security: Certification Issues and Technologies. *CrossTalk - The Journal of Defense Software Engineering*, 21(10), 2008.

[CL07] T J Cockram and S R Lautieri. Combining Security and Safety Principles in Practice. In *2nd Institution of Engineering and Technology International Conference on System Safety.* IEEE, 2007.

[Cor99] Pierre Corneillie. End of Project Report. Technical report, ACTS FP4 Program, 1999.

[DKV97] Fonda Daniels, Kalhee Kim, and Mladen A Vouk. The Reliable Hybrid Pattern A Generalized Software Fault Tolerant Design Pattern. In *European Conference on Pattern Language of Programs (EuroPLoP)*, pages 1–9, 1997.

[DL06] Brian Dobbing and Samantha Lautieri. SafSec : Integration of Safety & Security Certification. Technical report, Praxis High Integrity Systems, 2006.

[Dou98] Bruce Powel Douglass. Safety-Critical Systems Design. *Electronic Engineering*, 70(862), 1998.

[Dou02] Bruce Powel Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems.* Pearson, 2002.

[Dou10] Bruce Powel Douglass. *Design Patterns for Embedded Systems in C.* Elsevier, 2010.

[EPSW10] Christian Eherer, Henrik J Putzer, Franz Strasser, and Marko Wolf. Synergetic Safety and Security Engineering Improving Efficiency and Dependability. In *8th Embedded Security in Cars Workshop*, Bremen, Germany, 2010.

[Eur06] European Parliament. Directive 2006/42/EC of the European Parliament and of the Council, vol. L 157/24, 2006.

[Fan13] Pasquale Fanelli. IEC 61508 and IEC 61511: application state and trends. In *4th International Symposium on Loss Prevention and Safety Promotion in the Process Industries*, 2013.

[FGG+05] Ingmar Fliege, Alexander Geraldy, Reinhard Gotzhein, Thomas Kuhn, and Christian Webel. Developing safety-critical real-time systems with SDL design patterns and components. *Computer Networks*, 49(5):689–706, 2005.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, 1994.

[GJØO10] T.O. Grøtan, M.G. Jaatun, K. Øien, and T. Onshus. The sesa method for assessing secure remote access to safety instrumented systems, technical report a1626. Technical report, SINTEF, 2010.

[GLW12] John Goodenough, Howard Lipson, and Chuck Weinstock. Arguing Security - Creating Security Assurance Cases. Technical report, US Department of Homeland Security, 2012.

[Gru03] Lars Grunske. Transformational Patterns for the Improvement of Safety Properties in Architectural Specification. In *Nordic Conference on Pattern Languages of Programs (VikingPLoP)*, 2003.

[GSN11] GSN Community. GSN Community Standard, Verison 1, 2011.

[Ham12] Paul Hampton. Survey of safety Architectural Patterns. In *Achieving Systems Safety*, pages 7–9. Springer London, London, 2012.

[Han07] Robert S. Hanmer. *Patterns for Fault Tolerant Software.* Wiley, 2007.

[Har11] Neil Harrison. *Improving Quality Attributes of Software Systems Through Software Architecture Patterns.* Phd thesis, University of Groningen, 2011.

[HCS04] S Halkidis, Alexander Chatzigeorgiou, and George Stephanides. A qualitative evaluation of security patterns. In *6th International Conference on Information and Communications Security*, pages 132–144. Springer, 2004.

[HCS06a] Spyros T. Halkidis, Alexander Chatzigeorgiou, and George Stephanides. A qualitative analysis of software security patterns. *Computers & Security*, 25(5):379–392, July 2006.

[HCS06b] Spyros T Halkidis, Alexander Chatzigeorgiou, and George Stephanides. Quantitative Evaluation of Systems with Security Patterns Using a Fuzzy Approach. In *roceedings of the 2006 international conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET*, pages 554–564. Springer, 2006.

[HDGJ10] Brahim Hamid, Nicolas Desnos, Cyril Grepet, and Christophe Jouvray. Model-based security and dependability patterns in RCES - the TERESA Approach. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems - S&D4RCES '10.* ACM Press, 2010.

[HL03] M. Howard and D. LeBlanc. *Writing Secure Code.* Microsoft Press, 2003.

[HS13] André Alexandersen Hauge and Ketil Stølen. Developing Safe Control Systems using Patterns for Assurance. In *The Third International Conference on Performance, Safety and Robustness in Complex Systems and Applications.* IARIA, 2013.

[HTCS08] ST Halkidis, Nikolaos Tsantalis, A. Chatzigeorgiou, and G. Stephanides. Architectural Risk Analysis of Software Systems Based on Security Patterns. *IEEE Transactions on Dependable and Secure Computing*, 5(3):129–142, July 2008.

[IJ13] Azianti Ismail and Won Jung. Research Trends in Automotive Functional Safety. In *Proceedings of the International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE).* IEEE, 2013.

[Int02] International Electrotechnical Commission. IEC 62278, Railway Applications - The Specification and Demonstration of Reliability, Availability, maintainability and Safety (RAMS), 2002.

[Int10] International Electrotechnical Commission. IEC 61508, Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety Related Systems, 2010.

[Int14a] International Electrotechnical Commission. IEC 60601-1, Medical electrical equipment - Part 1: General requirements for basic safety and essential performance, 2014.

[Int14b] International Electrotechnical Commission. IEC 61513, Nuclear power plants - Instrumentation and control important to safety - General requirements for systems, 2014.

[Int14c] International Electrotechnical Commission. IEC 62061, Safety of machinery: Functional safety of electrical, electronic and programmable electronic control systems, 2014.

[Joh11] Chris W Johnson. Using Assurance Cases and Boolean Logic Driven Markov Processes to Formalise Cyber Security Concerns for Safety-Critical Interaction with Global Navigation Satellite Systems. In *Fourth International Workshop on Formal Methods for Interactive Systems*, 2011.

[JY11a] C W Johnson and A Atencia Yepez. Cyber Security Threats to Safety-Critical Space-Based Infrastructures. In *Proceedings of the Fifth Conference of the International Association for the Advancement of Space Safety*, 2011.

[JY11b] Chris W. Johnson and A. Atencia Yepez. Mapping the Impact of Security Threats on Safety-Critical Global Navigation Satellite Systems. In *Proceedings of the 29th International Systems Safety Society*. International Systems Safety Society, 2011.

[KM98] Tim Kelly and John McDermid. Safety Case Patterns - Reusing Successful Arguments. In *Colloquium on Understanding Patterns and Their Application to Systems Engineering*, 1998.

[KP10a] Kiran Kumar and T V Prabhakar. Pattern-oriented Knowledge Model for Architecture Design. In *17th Conference on Pattern Languages of Programs (PLoP)*, 2010.

[KP10b] Kiran Kumar and T.V. Prabhakar. Design Decision Topology Model for Pattern Relationship Analysis. In *1st Asian Conference on Pattern Languages of Programs (AsianPLoP 2010)*, 2010.

[KSB$^+$04] C. Kehren, C. Seguin, P. Bieber, C. Castel, C. Bougnol, J.P. Heckmann, and S. Metge. Architecture patterns for safe design. In *AAAF 1st Complex and Safe Systems Engineering Conference (CS2E)*, 2004.

[Lan11] R. Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security Privacy, IEEE*, 9(3):49 –51, 2011.

[LCJ05] Samantha Lautieri, David Cooper, and David Jackson. SafSec: Commonalities Between Safety and Security Assurance. *Constituents of Modern System-safety Thinking*, (February):65–75, 2005.

[Mel13] Eline Marie Bye Melkild. Goal and evidence based dependability assessment. Technical report, Norwegian University of Science and Technology, 2013.

[MFMP10] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. A comparison of software design security metrics. In *Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA '10*. ACM Press, 2010.

[MME$^+$13] Silvana Togneri MacMahon, Fergal McCaffery, Sherman Eagles, Frank Keenan, Marion Lepmets, and Alain Renault. Assessing against iec 80001-1. In *HEALTHINF*, pages 305–308. SciTePress, 2013.

[MR12] Bill Miller and Dale Rowe. A Survey SCADA of and Critical Infrastructure Incidents. In *Proceedings of the 1st Annual Conference on Research in Information Technology (RIIT)*. ACM Press, 2012.

[NMD09] Igor Nai Fovino, Marcelo Masera, and Alessio De Cian. Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety*, 94(9):1394–1402, September 2009.

[Nor09] Odd Nordland. EWICS TC7 - an insider's tip for safety and security guidance. *Newsletter of the UK's Safety-Critical Systems Club*, 18(2), 2009.

[NVSB12] Sunil Nair, Jose Luis la De Vara, Mehrdad Sabetzadeh, and Lionel Briand. An Extended Systematic Literature Review on Provision of Evidence for Safety Certification. Technical report, Simula Research Laboratory, 2012.

[NWD$^+$12] A. Nicholson, S. Webber, S. Dyer, T. Patel, and H. Janicke. SCADA Security in the Light of Cyber-Warfare. *Computers & Security*, 31(4):418–436, 2012.

[OPF13] Luiza Ocheana, Dan Popescu, and Gheorghe Florea. Integrating versus interfacing safety and security with process control system. In *Proceedings of the 19th International Conference on Control Systems and Computer Science*. IARIA, 2013.

[PCB09] Ludovic Piètre-Cambacédès and Marc Bouissou. The promising potential of the BDMP formalism for security modeling. In *Supplemental volume of the Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009)*. IEEE, 2009.

[PCB10a] Ludovic Piètre-Cambacédès and Marc Bouissou. Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP). In *European Dependable Computing Conference*, pages 199–208. IEEE, 2010.

[PCB10b] Ludovic Piètre-Cambacédès and Marc Bouissou. Modeling safety and security interdependencies with BDMP (Boolean logic Driven Markov Processes). In *International Conference on Systems, Man and Cybernetics*, pages 2852–2861. IEEE, 2010.

[PCQH13] L. Pietre-Cambacedes, E. Quinn, and L. Hardin. Cyber Security of Nuclear Instrumentation & Control Systems: Overview of the IEC Standardization Activities. In *Proceedings of the 7th IFAC Conference on Manufacturing Modelling, Management, and Control.* International Federation of Automatic Control, 2013.

[PKCD11] Zlatko Petrov, Kamil Kratky, Joao M P Cardoso, and Pedro C Diniz. Programming safety requirements in the REFLECT design flow. In *2011 9th IEEE International Conference on Industrial Informatics*, pages 841–847. IEEE, 2011.

[PKK12] Christopher Preschern, Nermin Kajtazovic, and Christian Kreiner. An architecture for safe and secure automation system devices and maintenance process. In *Proceedings of the 19th International Conference and Workshops on Engineering of Computer-Based Systems (ECBS)*. IEEE, 2012.

[PR13] Roberto Pietrantuono and Stefano Russo. Introduction to safety critical systems. In *Innovative Technologies for Dependable OTS-Based Critical Systems.* Springer, 20013.

[Pul01] L. Pullum. *Software fault tolerance techniques and implementation.* Artech House, 2001.

[RHFP13] Ansgar Radermacher, Brahim Hamid, Manel Fredj, and Jean-Louis Profizi. S&D Patterns for Component-Based Applications with Safety Requirements. In *European Conference on Pattern Language of Programs (EuroPLoP)*, 2013.

[RK13] Jari Rauhamäki and Seppo Kuikka. Patterns for control system safety. In *18th European Conference on Pattern Languages of Programs (EuroPLoP)*, 2013.

[RLRP13] D.C. Rispoli, M. Lourdes, V.C. Rispoli, and Fernandes P.G. Software Lifecycle Activities to Improve Security Into Medical Device Applications. In *Proceedings of the Sixth International Conference on Advances in Computer-Human Interactions.* IARIA, 2013.

[RTC12] RTCA, Incorporated. DO-178C, Software Considerations in Airborne Systems and Equipment Certification, 2012.

[RVK12] Jari Rauhamäki, Timo Vepsäläinen, and Seppo Kuikka. Architectural patterns for functional safety. In *Nordic Conference on Pattern Languages of Programs (VikingPLoP)*, 2012.

[RVK13a] Jari Rauhamäki, Timo Vepsäläinen, and Seppo Kuikka. Patterns for safety and control system cooperation. In *Nordic Conference on Pattern Languages of Programs (VikingPLoP)*, 2013.

[RVK13b] Jari Rauhamäki, Timo Vepsäläinen, and Seppo Kuikka. Patterns in Safety System Development. In *PESARO 2013, The Third International Conference on Performance, Safety and Robustness in Complex Systems and Applications*, pages 9–15. IARIA, 2013.

[Sar02] Titos Saridakis. A System of Patterns for Fault Tolerance. In *EuroPLoP*, 2002.

[SB12] Andreas Schaad and Mike Borozdin. TAM2: Automated Threat Analysis. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1103–1108. ACM, 2012.

[SBMS10] R.M. Suresh-Babu, U Mahapatra, and G.P. Srivastava. I&C Security for Nuclear Power Plants: A Study. *Bhabha Atomic Research Center Newsletter*, 2010.

[SL13] Nuno Silva and Rui Lopes. Practical Experiences with real-world systems: Security in the World of Reliable and Safe Systems. In *Proceedings of the 43rd Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2013.

[SLS$^+$13] Max Steiner, Peter Liggesmeyer, A G Software, Engineering Dependability, and T U Kaiserslautern. Finding Security Problems That Threaten The Safety of a System Analysis Process. In *International Conference on Computer Safety, Reliability and Security (SAFECOMP)*. Springer, 2013.

[SQU99] SQUALE Consortium. SQUALE Dependability Assessment Criteria, 4th draft, 1999.

[SS04] David Smith and Kenneth Simpson. *Functional Safety. A Straightforward Guide to Applying IEC 61508 and Related Standards.* Elsevier / Butterworth-Heinemann, 2004.

[SSS09] L. Schnieder, E. Schnieder, and T. Stnder. Research Trends in Automotive Functional Safety. In *Proceedings of the International Railway Safety Conference 2009*, 2009.

[TAFR02] Carol Taylor, Jim Alves-Foss, and Bob Rinker. Merging Safety and Assurance: the Process of Dual Certification of Software. In *Proceedings of the Software Technology Conference*, 2002.

[Tem11] S.J. Templeton. Security Aspects of Cyber-Physical Device Safety in Assistive Environments. In *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assisted Environments (PETRA)*, 2011.

[US 12] US Department of Defense. MIL-STD-882E: Department of Defense Standard Practice System Safety, 2012.

[VM01] John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley, 2001.

[Wu03] Weihang Wu. Safety Tactics for Software Architecture Design, 2003.

[Wu07] Weihang Wu. *Architectural Reasoning for Safety-Critical Software Applications.* PhD thesis, University of York, 2007.

[YS08] Artsiom Yautsiukhin and Riccardo Scandariato. Towards a quantitative assessment of security in software architectures. In *13th Nordic Workshop on Secure IT Systems (NordSec)*, 2008.