Deliverable D6.4

Mitigation Identification and Design

Technical References

Document version	:	1.0
Submission Date	:	31/12/2021
Dissemination Level	:	Public
Contribution to	:	WP6 – Risk Analysis and Mitigation Strategies
Document Owner	:	Fraunhofer IESE
File Name	:	BIECO_D6.4_31.12.2021_V1.0
Revision	:	3.0

Project Acronym	:	BIECO
Project Title	:	Building Trust in Ecosystem and Ecosystem Components
Grant Agreement n.	:	952702
Call	:	H2020-SU-ICT-2018-2020
Project Duration	:	36 months, from 01/09/2020 to 31/08/2023
Website	:	https://www.bieco.org

This project has received funding from the European Union´s Horizon 2020 Research and Innovation Programme under Grand agreement No. 952702.

Revision History

REVISION	DATE	INVOLVED PARTNERS	DESCRIPTION
0.1	01.11.2021	IESE	Initial draft
0.2	12.11.2021	UMU	Section 5.4
0.2	15.11.2021	CNR	Section 2.7 (preliminary)
0.3	18.11. 2021	CNR	Section 3
0.4	22.11. 2021	IESE	Section 2.5 on Goals structuring in Trust-based Digital Ecosystems
0.4	22.11. 2021	UMU	Section 7 on stricter MUD application. Acronyms Table.
0.5	23.11. 2021	IESE (on behalf of CNR)	Changes from CNR being consolidated
0.6	24.11. 2021	IESE	Content for sections 4, 5, 6, and 7 added
0.7	24.11. 2021	IESE	Most content has been consolidated across all sections; included Appendix from CNR
0.8	25.11. 2021	IESE	Correcting formatting, correcting references
0.9	25.11. 2021	UMU	Corrections, refinement, and example added to section 7.1
0.9	26.11. 2021	RES	Internal review
1.0	29.11. 2021	IESE	Minor corrections, references, input from CNR consolidated
1.1	29.11. 2021	IESE	Document ready for BIECO internal review
1.2	15.12. 2021	UNI	Internal Review
1.3	16.12. 2021	GRAD	Internal Review
1.4	20.12. 2021	CNR	Response to Internal Reviews
2.0	27.12. 2021	IESE	Consolidation and final changes
2.1	29.12. 2021	UNI	Final Revision and correction by Coordinator
3.0	30.12.2021	UNI	Finalizing deliverable and submission

List of Contributors

Contributor(s): Ioannis Sorokos (IESE); Emilia Cioroaica (IESE); Adrián Sánchez (UMU); Felicita Di Giandomenico (CNR); Giulio Masetti (CNR) ; Sara N. Matheu (UMU) ; Eda Marchetti (CNR) ; Silvano Chiaradonna (CNR) ; Enrico Schiavone (RES)

Reviewer(s): Fillipa Ferrada (UNI); Nora M. Villanueva (GRAD); Sanaz Nikghadam-Hojjati(UNI); José Barata(UNI)

BIECO project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **952702**.

Acronyms

Acronym	Term
ICT	Information Communication Technology
HARA	Hazard Analysis and Risk Assessment
FTA	Fault Tree Analysis
FMEA	Failure Mode and Effects Analysis
TARA	Threat Analysis and Risk Assessment
ATA	Attack Tree Analysis
DDI	Digital Dependability Identity
ODE	Open Dependability Exchange (Metamodel)
GSN	Goal Structuring Notation
ACP	Assurance Claim Point
CAE	Claims Arguments Evidence
SACM	Structured Assurance Case Metamodel
MUD	Manufacturer Usage Description
ConSerts	Conditional Safety Certificates
DRM	Dynamic Risk Management
ITC	Intrusion Tolerance Countermeasure
CS	Constituent System
SG	Safety Guarantee
SD	Safety Demand
RtE	Runtime Evidence
DER	Distributed Energy Resource

Executive Summary

In this deliverable, we detail the BIECO approach towards assuring that safety, security, and other ICT risks have been adequately mitigated and controlled during development. The approach leverages guidelines based on international safety and security standards (e.g., ISO 26262 and ISO 21434). We further build upon this foundation by incorporating other BIECO contributions, in terms of methods (e.g., Attack Tree Analysis), models (predictive models and extended MUD files), and tools (ResilBlockly, safeTbox). Our contributed advancement is in operationalizing an integrated safety-security co-analysis process to yield risk mitigation assurance artefacts.

Project Summary

Nowadays most of the ICT solutions developed by companies require the integration or collaboration with other ICT components, which are typically developed by third parties. Even though this kind of procedures are key in order to maintain productivity and competitiveness, the fragmentation of the supply chain can pose a high security risk, as in most of the cases there is no way to verify if these other solutions have vulnerabilities or if they have been built considering the best security practices.

In order to deal with these issues, it is important that companies make a change on their mindset, assuming an “untrusted by default” position. According to a recent study [1] only 29% of IT business know that their ecosystem partners are compliant and resilient with regard to security. However, cybersecurity attacks have a high economic impact and it is not enough to rely only on trust. ICT components need to be able to provide verifiable guarantees regarding their security and privacy properties. It is also imperative to detect more accurately vulnerabilities from ICT components and understand how they can propagate over the supply chain and impact on ICT ecosystems. However, it is well known that most of the vulnerabilities can remain undetected for years, so it is necessary to provide advanced tools for guaranteeing resilience and also better mitigation strategies, as cybersecurity incidents will happen. Finally, it is necessary to expand the horizons of the current risk assessment and auditing processes, taking into account a much wider threat landscape. BIECO is a holistic framework that will provide these mechanisms in order to help companies to understand and manage the cybersecurity risks and threats they are subject to when they become part of the ICT supply chain. The framework, composed by a set of tools and methodologies, will address the challenges related to vulnerability management, resilience, and auditing of complex systems.

Partners


Disclaimer

Table of Contents

List of Figures

List of Tables

1. Introduction

In systems where dependability [2] (e.g., safety and/or security) concerns are critical, and absolute proof of dependability is not feasible due to the underlying system and its operational environment's complexity, the established approach is to argue that overall risk has been reduced to acceptable levels before the system is deployed into operation. Such arguments rely on the mitigation of dependability risk via rigorous development processes.

In more general terms, such approaches to controlling risk are also employed in domains where systems are providing 'mission-critical' services instead of strictly dependability-critical ones. For instance, while failure to deliver power to critical infrastructure (e.g., hospitals, residential areas in adverse weather) may not be directly safety-critical in the short-term (e.g., the presence of back-up power protects against immediate danger), it is certainly mission-critical in terms macroscopic and long-term consideration.

ICT systems are now ubiquitous across critical infrastructure, and while they may not always have direct dependability implications, as highlighted in the earlier paragraph, the risk they contribute to the overall infrastructure must also be assured to be acceptably controlled. The emergence of ecosystem concepts such as smart grids [3] and smart cities [4], and the ongoing advancement of the IoT paradigm [5] are further indicators of the need for developing dependable ICT backbones.

In this deliverable, we present the BIECO approach towards the definition of appropriate safety and security risk mitigation. The aim of this approach is to assure that the residual risk present in dependability and mission-critical systems (or specific subsystems thereof) is deemed acceptable by the end of development. Our proposed approach is based on guidelines found in established functional safety standards and integrated with those from corresponding security standards.

Specifically, we demonstrate how safety and security risk assurance can be constructed to argue that risk has been adequately mitigated during development, following established guidelines from both domains. Furthermore, we discuss how dynamic mitigation of risk can be employed as part of the underlying system's operation at runtime.

Our approach links to the rest of BIECO by:

- Providing methods appropriate for systematically structuring safety (T6.2) and security assurance claims (WP7) as part of assurance cases.
- Incorporating the risk assessment (T6.2) and security analysis (T6.1) process to provide appropriate development-time evidence of risk mitigation.
- Developing more resilient systems through the concept of Intrusion-Tolerant Architectures (WP6), intended to mask the effect of attack-induced failures, and integrate the developed redundancy schemes in the risk management process (WP6).
- Links failure and trust prediction concepts with dynamic risk management (WP4).
- Links runtime risk management and resilient adaptation (WP4).

The remainder of the deliverable is structured as follows:

- Section 2 discusses some of the motivation behind the approach and provides relevant background including previous and related work.
- Section 3 provides key considerations regarding the choice and application of intrusion-tolerance countermeasures.

- Section 4 addresses the overall issue of dependability and security risk mitigation during system development as part of BIECO.
- Section 5 presents a combined dependability and security approach for managing risk at runtime.
- Section 6 discusses tool support for development and runtime dependability risk mitigation.
- Section 7 provides a simplified example of how the tool support mentioned in Section 6 can be used.
- Section 8 provides a summary of the deliverable.

2. Motivation & Background

The BIECO project use cases involve ICT infrastructure that may be individual or constituent systems of larger dependability or mission-critical systems. For example, ICT gateways usually may not have direct catastrophic safety effects in case of failure or attack, but if they are part of a larger critical infrastructure e.g., a smart grid, then their role must be also considered accordingly. The implication of developing such systems is that they are part of a larger ecosystem of stakeholders, technologies, devices, and software, whose interactions need to be accounted for and managed appropriately.

When safety is a primary concern of a given application, then it stands to reason that critical safety hazards (i.e., negative events in terms of safety) should be identified early in the development process, so that important requirements are not discovered in later stages, where major changes require increasing development resources. As such, when safety issues are known, security, availability, reliability, and other related properties can be analysed more effectively, using the knowledge of their effects on the impacts of those safety issues.

That being said, safety is not always a primary concern; indeed, given the nature of ICT systems, security is the more typical aspect to be considered. In particular, unique security concerns such as privacy are usually orthogonal to safety and must be considered regardless. In other cases where mission-critical systems are involved, e.g., power production and distribution, availability of the overall system (e.g. continuous provision of power, despite failures and/or attacks) may be a primary objective on its own. Our approach should therefore be flexible in addressing dependability aspects interchangeably, depending on the application system being considered.

In both the security and safety domains, standardization is extensive, and enables us to draw from established best-practice guidelines applied in industry. Functional safety standards, such as IEC 61508 (general) and ISO 26262 (automotive), provide guidance for identifying and managing safety risk, while maintaining adequate integrity of the development process. On the security side, standards such as ISO 21434 provide corresponding guidelines for addressing security risk for commercial vehicle development.

Both ISO 26262 and 21434 include provisions for developing system elements ‘out of context’ (EooC), which offer a useful development paradigm for constituent systems and components that are developed asynchronously from the rest of their encompassing system. Using this paradigm, development of ICT systems (e.g. gateways) can be encapsulated using clearly specified interfaces, in terms of both functionalities, as well as requirements.

To address dependability risk in complex systems, we can argue that the development process has identified all relevant risks through sound analysis, designed and implemented appropriate means of mitigating (or eliminating) these risks, and confirmed that the above process has acceptably reduced the residual risk by the end of development. Capturing such arguments in ad-hoc natural language has proven inefficient and error-prone [6], and modern systems are large and complex enough to require significant argumentation and documentation. Assurance cases structure clear, comprehensive, and convincing arguments that the underlying system will operate dependably (to a degree acceptable by stakeholders) [7]. These arguments typically establish some high-level claims regarding desired dependability properties of the system, decomposing them along strategies of argumentation, eventually down to specific evidence of system design, implementation, and/or dependability-related analysis, produced over the course of system development. Given their flexibility, they

are an appropriate medium for communicating how risk is mitigated, regardless of whether the risk is related to safety, security, or other mission-critical aspects.

Typically, due to the uncertainty regarding the exact conditions of the operational situation the system will experience at runtime, worst-case assumptions with regards to the assessment and assurance of risk are established. The implication that such assumptions have is that they often deactivate functionality or severely restrict its performance in order to manage the impact of risk in worst-case conditions.

Our approach intends to improve upon this limitation by extending the risk assurance process to account for operational conditions and adjust functionality, accordingly, thereby avoiding worst-case assumptions when feasible. The models which enable this extension are known as Conditional Safety Certificates (ConSerts). ConSerts capture internal and external variability of the underlying system, and specify how it can adapt, given its observed environment and relied-upon systems.

Both construction of integrated assurance case and ConSerts rely on combining risk assurance models with dependability analysis techniques and models, which provide the concrete evidence for arguing risk and assessing which adaptation is needed. Both the assurance models, as well as the evidentiary models are often heterogeneous, both in technique and tool chosen, meaning common metamodels that can integrate them are needed. For our approach, we leverage the past experience from the AMADEOS and DEIS projects and use the corresponding metamodels they produced.

As an example of how a security-specific concern can be addressed as part of our flexible approach, we discuss its application for developing intrusion-tolerant systems. In such systems, while there may be other defenses to prevent intrusions, there is also a need to handle the continuous operation even in the presence of a successful intrusion. System architectures featuring Intrusion-Tolerant Countermeasures (ITC) offer answers in this regard, enabling the underlying system to robustly adjust its operation and continue providing a dependable service. We discuss how our approach can integrate ITC architectures in terms of assurance argumentation and ConSerts in 5.2.

As our approach needs to be practically applicable towards large and complex systems, there is a need for appropriate model-based tooling. Such tools can assist development by managing complexity, reducing effort and manual errors, and integrate with the rest of the BIECO through the artifacts and services the produce and consume.

In the remainder of this section, we provide a brief overview regarding the status quo on the following topics; in 2.1 and 2.2, we provide a brief summary of the development lifecycle according to the safety and security standards ISO 26262 and ISO 21434 respectively. Section 2.3 addresses existing views on safety and security co-analysis. In 2.4, we discuss assurance cases and the associated notation systems. Goals in ecosystems are then discussed in section 2.5. In 2.6, we present existing work on ConSerts. In 2.7, we discuss the concept of the Digital Dependability Identity (DDI), which can be used for exchanging model-based dependability information. Finally, in 2.8, we discuss the concept of ITCs and set the basis for a conceptual framework to develop redundancy-based solutions to mask the effect of attack-induced failures.

2.1. Functional Safety Standard Development Lifecycle

The International Electrotechnical Commission (IEC) standard 61508 addresses functional safety of Electric/Electronic/Programmable-Electronic (E/E/PE) (safety-related) systems. Functional safety refers to the part of the overall safety (of the system under development) that is provided by active safety measures. For IEC 61508, this scope is further restricted to active safety measures implemented through E/E/PE technologies. Numerous domain-specific standards have been produced by extending the IEC 61508 concepts, and tailoring them for their corresponding domain e.g., the Society of Automotive Engineers (SAE) Aerospace Recommended Practice (ARP) 4754-A for aviation and the International Organization for Standardization (ISO) 26262 for automotive.

The IEC 61508 further introduces two important concepts that are shared with numerous other standards as well, namely a generic system development lifecycle, and Safety Integrity Levels (SILs). The generic lifecycle model is widely referred to as the 'V-model', an overview of which can be seen in Figure 1. In the figure, the 'nominal' phases are identified in blue, and the assurance-related activities in orange.

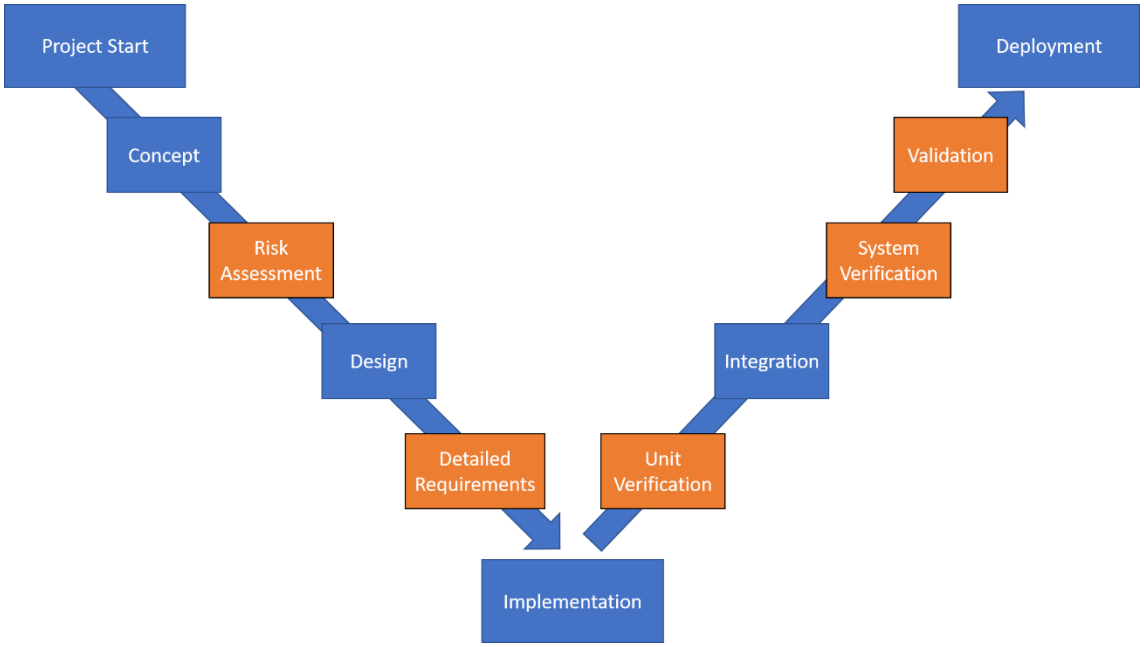


Figure 1 - IEC 61508 Lifecycle Overview

According to the lifecycle, at the outset of a system development project, the concept phase initiates. During the concept phase, the need and purpose of the underlying system is specified, along with relevant assumptions e.g., regarding its operational context. Once the high-level specification becomes more refined with functions, risk assessment of the nominal functionality can be initiated.

The process outlined in the standard is heavily risk-based, as the choice of safety measures, and the assurance activities associated with their development, depend on the underlying system's safety risk analysis. Such Hazard Analysis and Risk Assessment (HARA) processes are conducted by identifying hazards, which are events that, if they occur, may have negative effects in terms of safety. For each hazard, its contribution to the overall risk of the system can be evaluated in terms of the impact and severity of its effects, and the likelihood of its occurrence. Domain-specific factors and risk classification systems allow the process to be tailored to the particular characteristics of a given application. For example, ISO 26262 considers situational operations in more

detail e.g. driving on highways during rain. As a result of the HARA, from the set of hazards and/or hazardous situations, corresponding high-level safety goals can be specified; the system design through which they are satisfied is referred to as the safety concept.

SILs can be assigned to hazards, hazardous situations, and to the corresponding safety goals, according to their determined level of risk. Higher SILs indicate that the corresponding hazard(s) are of higher risk. By extension, the safety functions designed to mitigate higher SIL hazards must be associated with more stringent safety requirements.

The safety concept specifies the set of safety measures that satisfy the corresponding safety goals. Once functional safety requirements have been specified, and as the (architecture) design phase of development assigns the nominal (and safety) functions to concrete systems, concrete software and hardware requirements can be specified and allocated. SILs from the functional level can also be distributed across the constituent elements of the overall system i.e., lower-level subsystems and components.

As individual elements of the overall system are implemented, they are unit tested, according to their specifications and SILs. As SIL increases, more rigorous testing techniques and criteria are recommended, e.g. fault injection [8] [9] and Modified Condition/Decision Coverage (MC/DC) [10]. Assuming successful unit validation, (sub)system verification iterates alongside the integration process as larger subsystems are composed. Eventually, the functional safety requirements of the safety concepts, and their corresponding safety goals, are validated, ideally confirming that the residual risk across all hazards is acceptably low.

Different kinds of cause-effect analyses allow higher-level goals/effects (e.g. protection against specific hazard) to be linked to more detailed requirements/causes, qualitatively ('which failures can cause a hazard') or quantitatively ('how likely are failures to cause a hazard'), and can be performed inductively (bottom-up) or deductively (top-down). Typical examples include Fault Tree Analysis (FTA) [11], and Failure Modes and Effects Analysis (FMEA) [12].

In many domains, and particularly in the automotive domain, systems are typically developed in a distributed flow, either by outsourcing parts of development, or by procuring commercial 'off-the-shelf' components. In either case, the elements provided by the external supplier must be validated carefully before being integrated into the system being developed. Element-out-of-Context (EooC) development formalizes this development paradigm by establishing a clear interface between the element supplier and the system manufacturer.

Under EooC development, suppliers establish at the outset assumptions regarding the operational environment of the host system and derive corresponding requirements under which the EooC can deliver its specified functionality. Effectively, this means specifying functional safety requirements that are assumed will be present in the host system. Development of the EooC can then proceed following the development lifecycle processes outlined above, eventually validating the implemented EooC against the assumptions and requirements of the presumed operational environment. Once the EooC development completes, it can be evaluated and integrated into its host system's development lifecycle accordingly.

In parallel to the rest of the standard activities outlined above (and others not explicitly mentioned here as well), a safety case is also prescribed for documenting the rationale of the choices made regarding identification and management of safety-related risk, and

documenting the activities and work products related to the standard. More information regarding such cases is provided in Section 2.4.

2.2. Security Development Lifecycle

ISO 21434 is an automotive cybersecurity standard, closely aligned with the ISO 26262 processes. As such, ISO 21434 activities build upon the V-model lifecycle, and some of the terminology is shared across the two standards.

ISO 21434 project-dependent activities begin at the point of specification of the system under development. Once the target functionality is clear (and a determination of what is cybersecurity-relevant is made), a Threat Analysis and Risk Assessment (TARA) can be applied. The role of the TARA is to:

- identify the critical aspects of the system that need to be protected against potential attacks (aka 'assets')
- identify the specific security properties of each asset that need to be protected
- assess the potential impact of each attack
- analyze the potential attack paths e.g., using Attack Tree Analysis (ATA) [13] or Failure Mode and Effects Vulnerability Analysis (FMEVA) [14]
- assess the feasibility of each attack
- assess the risk of each threat scenario by combining the potential impact and the feasibility of the attacks that realize it [15]

For each threat scenario whose risk is deemed significant, security goals are specified to mitigate the effects of its impact or reduce the feasibility of the attacks that can realize the scenario. As per the safety concept from ISO 26262, a security concept specifies how security goals are met, by establishing corresponding security controls. Security controls can then be used to specify security requirements that need to be implemented.

Finally, as per ISO 26262, developing elements out-of-context is also supported explicitly in ISO 21434, as is developing a security case, which is responsible for collecting argumentation regarding the choices made over the course of the security assurance, and documentation of the corresponding activities and work products, including those mentioned earlier in this section. More information on assurance cases can be found in Section 2.4.

2.3. Safety(/Dependability) and Security Co-Assurance

When developing systems that have dependability (e.g., safety, security, …) or mission-critical characteristics (often availability-related), existing standards provide targeted guidelines towards corresponding aspects (e.g. ISO 26262 for safety and ISO 21434 for security). However, integrating crosscutting concerns efficiently and effectively is not straightforward. System developers need to navigate the legal and regulatory landscape which may introduce domain-specific restrictions, incorporate assurance methods from different domains, specify requirements that also correctly account for cross-domain interaction, arrive at an implementation that satisfies all above requirements and stakeholders, and was developed rapidly enough to yield a return on the invested resources.

A particular challenge of the above puzzle is the co-assurance of systems development in terms of both safety and security. As indicated in sections 2.1 and 2.2, standards exist for addressing safety and security individually, but they prescribe only partial guidelines for handling detailed interactions across their activities. For instance, in ISO 21434:2021, Annex D provides an example on determining whether a given element of the vehicle under development is cybersecurity-relevant or not. One of the conditions for considering an element relevant is whether the element contributes “to the safe operation of the vehicle” (ISO21434:2021, page 57). The implication here is that the determination of whether an element is safety-relevant, should happen at a stage where this knowledge is already available from the safety assurance activities e.g., during the concept phase of ISO 26262. Otherwise, if security analysis is applied beforehand, there is risk of needing to repeat parts of the security lifecycle, as elements revealed to be safety-critical need to be re-evaluated. Such scenarios could waste development effort in the worst case, as re-design could be needed.

Such inefficiency can be addressed in terms of safety-security co-assurance, e.g., in [16], the authors propose to synchronize the security lifecycle workflow such that a preliminary HARA has already been conducted and safety goals have been specified. At that point, safety goals can serve as assets to be protected in terms of security. An additional benefit is that the potential impact of security-related threats to those safety goals becomes clearer, and their risk can be more accurately evaluated.

In more general scenarios, where safety might not be relevant, this approach should be adapted by prioritizing the development workflow to assess dependability/mission-critical risks based on the priority of a given property for the application. For example, in a nuclear power production plant, safety is arguably the top-priority property, and its workflow should precede others'. In such cases, risks to safety should be identified early in the development lifecycle, so that other sources of risk (e.g., security) that could contribute to safety-related risks can be identified efficiently.

However, when considering systems where safety is not critical (e.g., an ICT gateway), other properties can be prioritized ahead of the security, if they are relevant. For example, if availability of a service provided by the ICT gateway is considered mission-critical, then identifying sources of risk against its availability can provide appropriate input for the security-specific TARA. This approach allows risks to be ‘triaged’, according to the application needs, while still supporting analytical coverage of all relevant risks. As an example of this mindset, see [17], where security risks are prioritized based on the criticality of the devices they can impact. This means that security-specific risks not directly contributing to other risks (e.g., privacy-related risks) can still be identified and addressed as part of the proposed process.

2.4. Assurance Cases

The safety, or more generally, assurance case, aims to provide a structured argument which should clearly, comprehensively, and correctly explain how the relevant risks of the element under discussion have been addressed as part of its development process [7]. The assurance case structures such arguments by establishing claims regarding the high-level properties that the element should satisfy, and eventually linking them with concrete evidence that supports the former. Graphical notations for representing such cases help to manage the complexity of such arguments and simplify their construction and management. The most well-known notations are the Goal Structuring Notation (GSN)[1] [7], Claims-Arguments-Evidence (CAE) [18], and the Structured Assurance Case Metamodel (SACM) [19]. Due to the existing support for GSN in the BIECO tool safeTbox[2], our approach is illustrated using said notation; however, the approach could be directly adapted to use the other notations as well, provided tool support for those is chosen or developed instead.

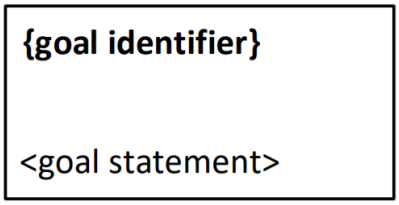
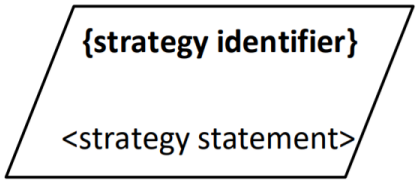
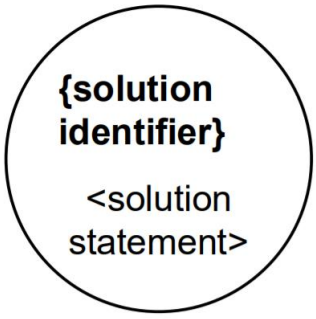
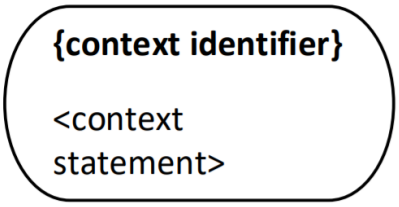
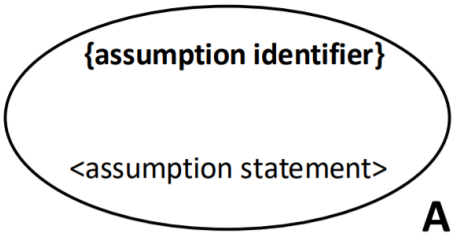
An assurance case is most appropriate to use when stakeholders of a complex system or process need to be convinced of its trustworthiness in detail. In such scenarios, focusing only on individual pieces or sets of evidence is not sufficient to provide confidence that all the relevant risks have been adequately addressed. A set of evidence can only be considered adequate if it is clearly contributing to specific claims addressing corresponding risks. Moreover, the assurance case must also convincingly argue that the processes which identified said risks and produced the cited evidence are also sound and correctly applied. Overall, the stakeholder must be convinced that all risks have been adequately addressed, and that the risks not explicitly addressed are acceptably low. In these terms, an assurance case generates stakeholder trust by clearly structuring and distinguishing between the evidence that was provided, but also the process, reasoning, and justification that led to the evidence creation.

Most assurance cases capture arguments mainly as part of the development-time activities. As such, these arguments address the relationship between the identified risk, safety goals and measures that mitigate them, and the verification and validation of the designed and implemented system. By expanding on the above notion of risk, assurance cases can flexibly be applied to address not only safety-related risk, but can also cover other dependability aspects e.g., risk to security, availability, maintainability, etc [20].

An overview of GSN elements can be seen in Table 1. Goal elements establish claims, usually regarding properties of the system that are proven as part of the assurance case, e.g., “system is acceptably dependable”. Strategy elements capture lines of reasoning which explain how Goals are supported by more detailed arguments, e.g., “argument of system safety via mitigation of risk across all safety hazards”. Solutions encapsulate references to specific pieces of evidence, usually produced as part of the development assurance activities. For example, to argue that all relevant hazards have been identified through appropriate means, a Solution citing the review of the risk analysis by an auditor can be included. Context, Assumption, and Justification elements function synonymously, providing relevant information regarding specific elements or the choice of argumentation. Goals, Strategies, and Solutions are typically linked via “Supported By” relationships, indicating how the subsequent elements support the former. Context, Assumptions, and Justifications are typically linked via “In Context Of” relationships.

[1] https://scsc.uk/gsn?page=gsn%202standard
[2] https://safetbox.de/

Table 1 - Overview of GSN Elements (from [21])

GSN Graphical Element Depiction	Definition
	A **goal**, presents a claim, part of an argument.
	A **strategy**, provides the inference between goals.
	A **solution**, references an evidence item.
	A **context**, references contextual information or statements.
	An **assumption**, presents an intentionally unsubstantiated statement.
	A **justification**, presents a statement of rationale.

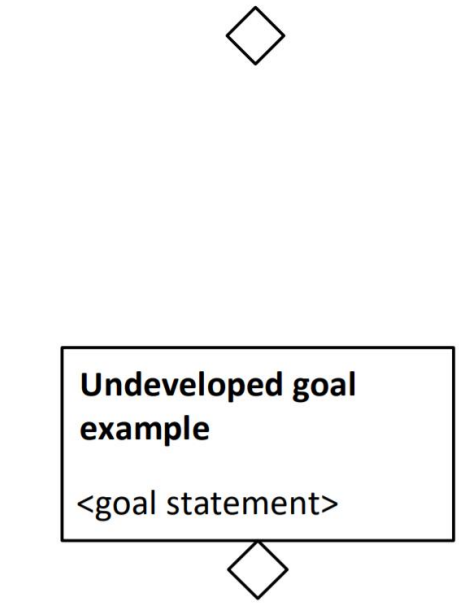
	An **undeveloped element decorator**, indicates a line of argument that has not been developed. Can apply to goals and strategies. For example, an **undeveloped goal**, presents a claim which is left undeveloped in the argument.
	**SupportedBy**, for goal-to-goal, goal-to-strategy, goal-to-solution, and strategy-to-goal relationships.
	**InContextOf,** for contextual relationships, e.g. goal-to-context, goal-to-assumption, goal-to-justification, strategy-to-context, etc.

Figure 2 shows a small abstract example of a GSN argument. Goal_182 establishes an initial claim, which in this case is that the system satisfies some abstract property P. As context to this claim, the description of the system and specification of the property can also be associated via Context_186 and Context_187. The latter can further point the reader to concrete documents outside of the GSN diagram. To support the claim, a line of reasoning is offered in Strategy_183, with an attached justification. Finally, proof of the property is claimed in Goal_184, and reference to the documentation serving as evidence is provided in Solution_185.

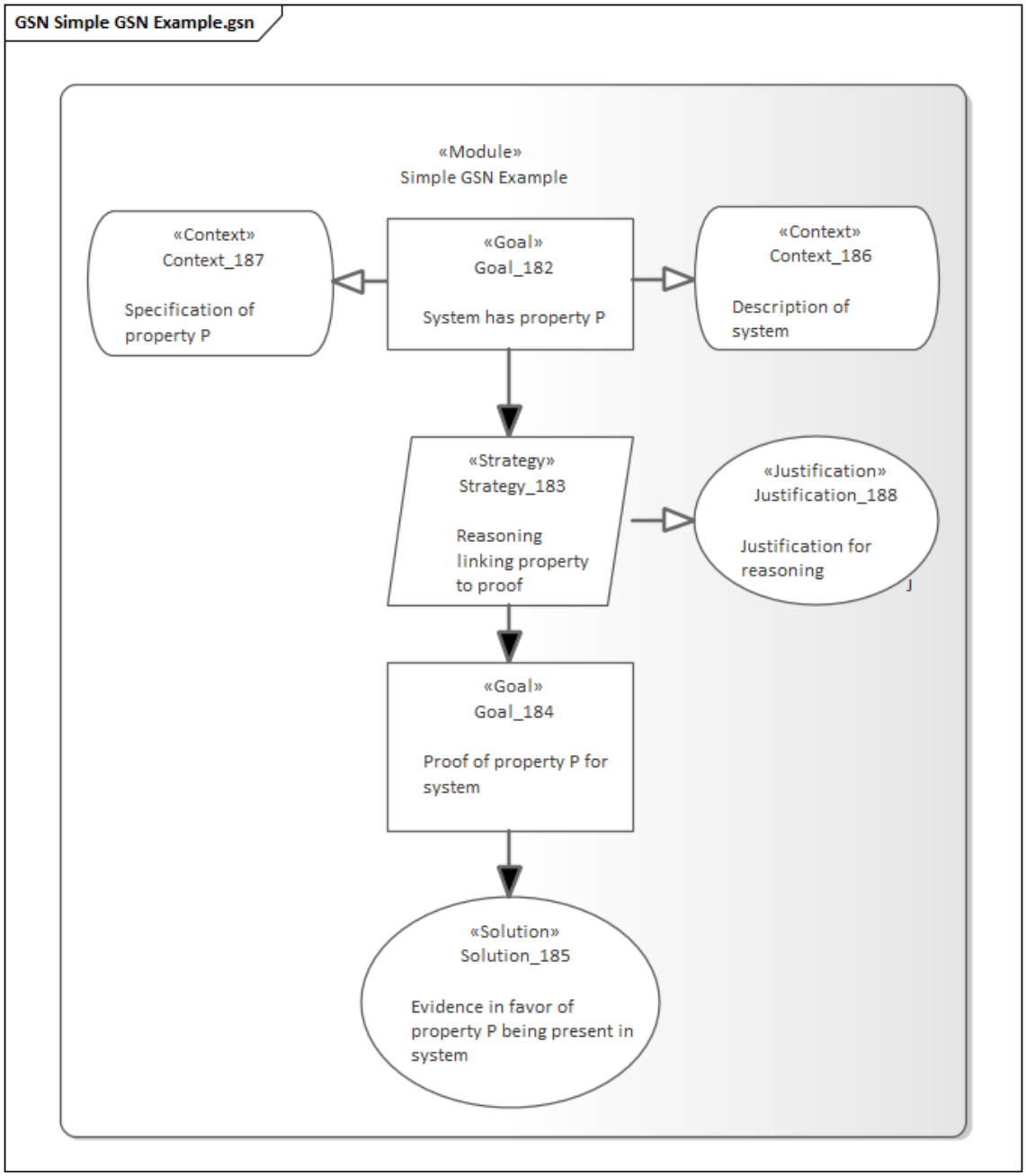
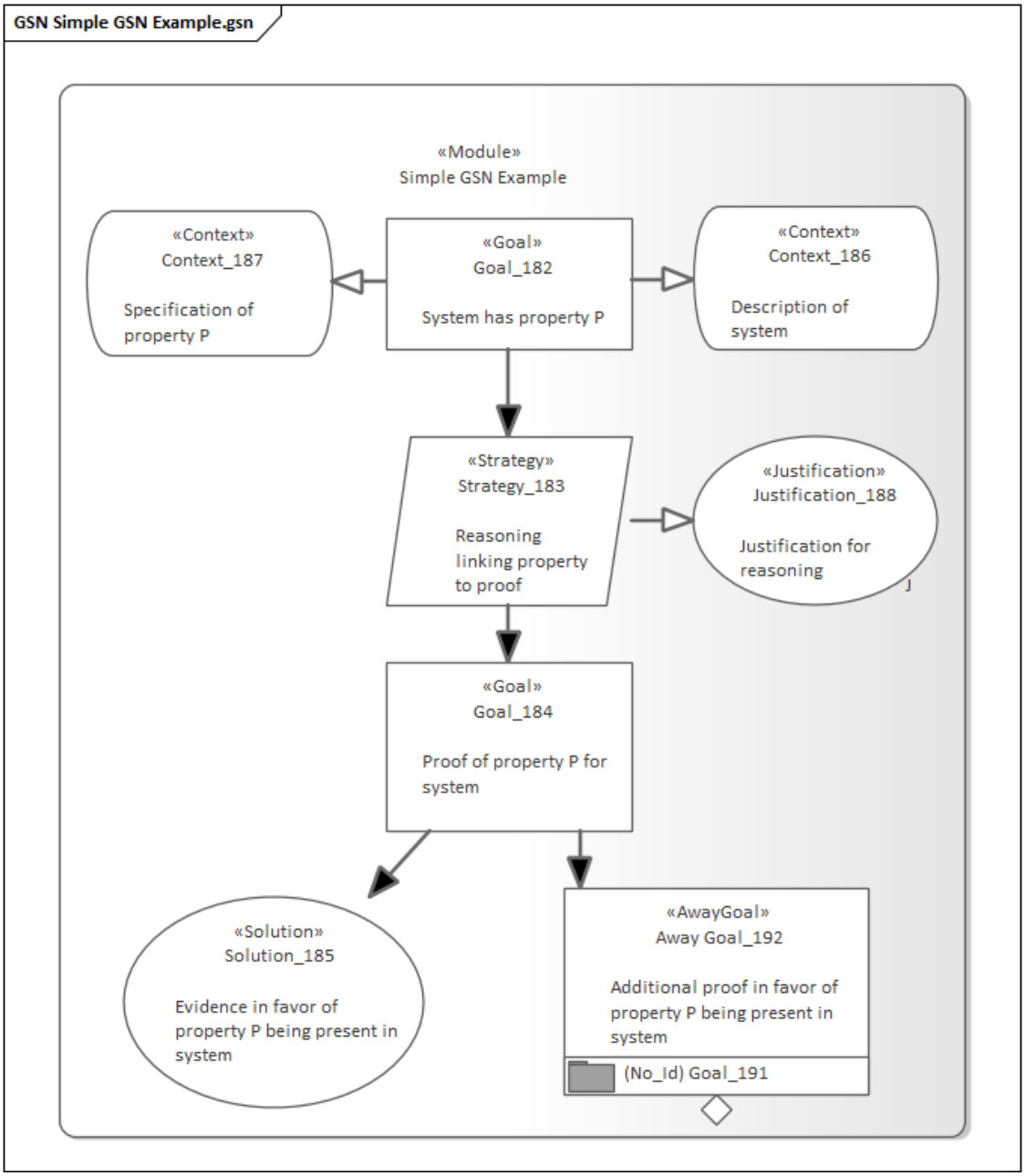


Figure 2 - Basic GSN Example

The 3rd version of the GSN community standard itself includes some additional notation concepts which will be briefly mentioned here as useful building blocks for structuring dependability assurance cases. These are assurance case modules, Assurance Claim Points (ACPs), and dialectic assurance case development.

Assurance case modules encapsulate parts of argumentation so that they can be isolated from the overall argument and cross-referenced as needed. This reduces the complexity of both reviewing and managing the structure of an assurance case. An example of GSN modules can be seen in Figure 3. The abstract from the example in Figure 2 has been extended, now including an additional supporting claim of proof of property P, through Away_Goal_192. Away_Goal_192 references an external goal, Goal_191, which is found in Module_190. Note that Goal_191 is also annotated as 'Undeveloped', indicating further refinement is needed.


GSN Simple GSN Example.gsn
«Module»
Simple GSN Example
«Context»
Context_187
Specification of property P
«Goal»
Goal_182
System has property P
«Context»
Context_186
Description of system
«Strategy»
Strategy_183
Reasoning linking property to proof
«Justification»
Justification_188
Justification for reasoning
«Goal»
Goal_184
Proof of property P for system
«Solution»
Solution_185
Evidence in favor of property P being present in system
«AwayGoal»
Away Goal_192
Additional proof in favor of property P being present in system
(No_Id) Goal_191

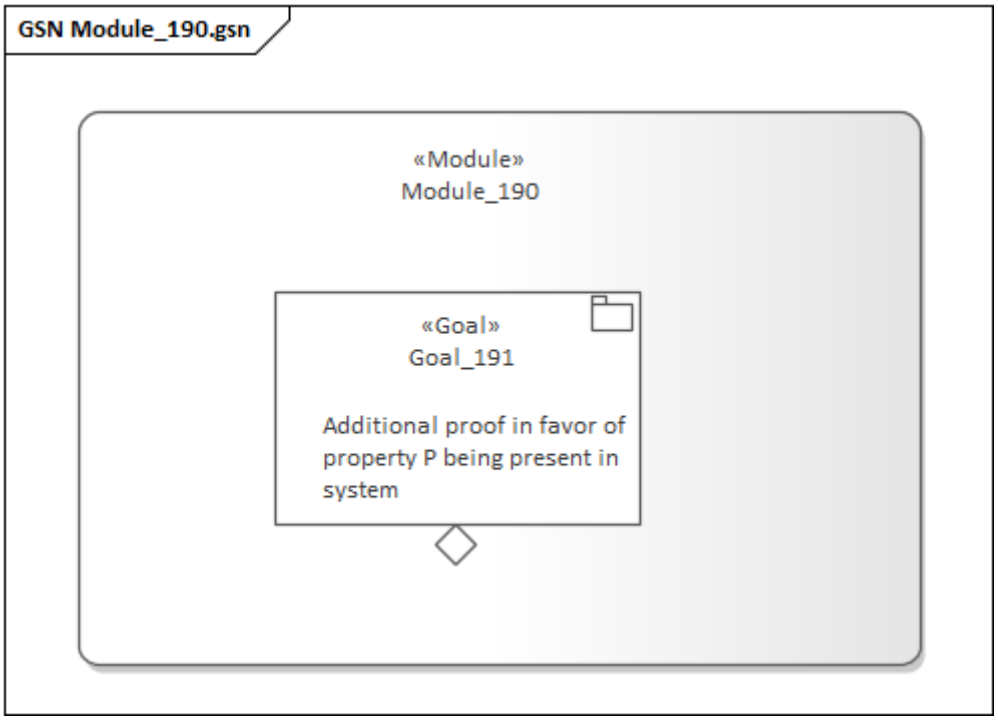


Figure 3 - Abstract GSN Module Example

ACPs are used to argue why the inclusion of elements or connectors in the assurance case is justified, embedding side-arguments on either the elements or the connectors linking said elements to the rest of the assurance case. ACPs allow confidence arguments to be seamlessly incorporated onto the main line of argument of an assurance case. An example of how ACPs can be applied is seen in Figure 4, where the example from Figure 3 has been further expanded to include ACPs annotated onto both elements and relationships.

For instance, ACP3 annotates the “In Context Of” relationship between Goal_182 and Context_186, which indicates that this relationship is further argued in the referenced Goal_191. In this example, one could argue that the system description referenced has been verified to be accurate, a side-argument that lends more credibility to the inclusion of the corresponding context element. Similarly, ACP4 and ACP5 indicate how a solution and justification element can also /be annotated with ACPs.

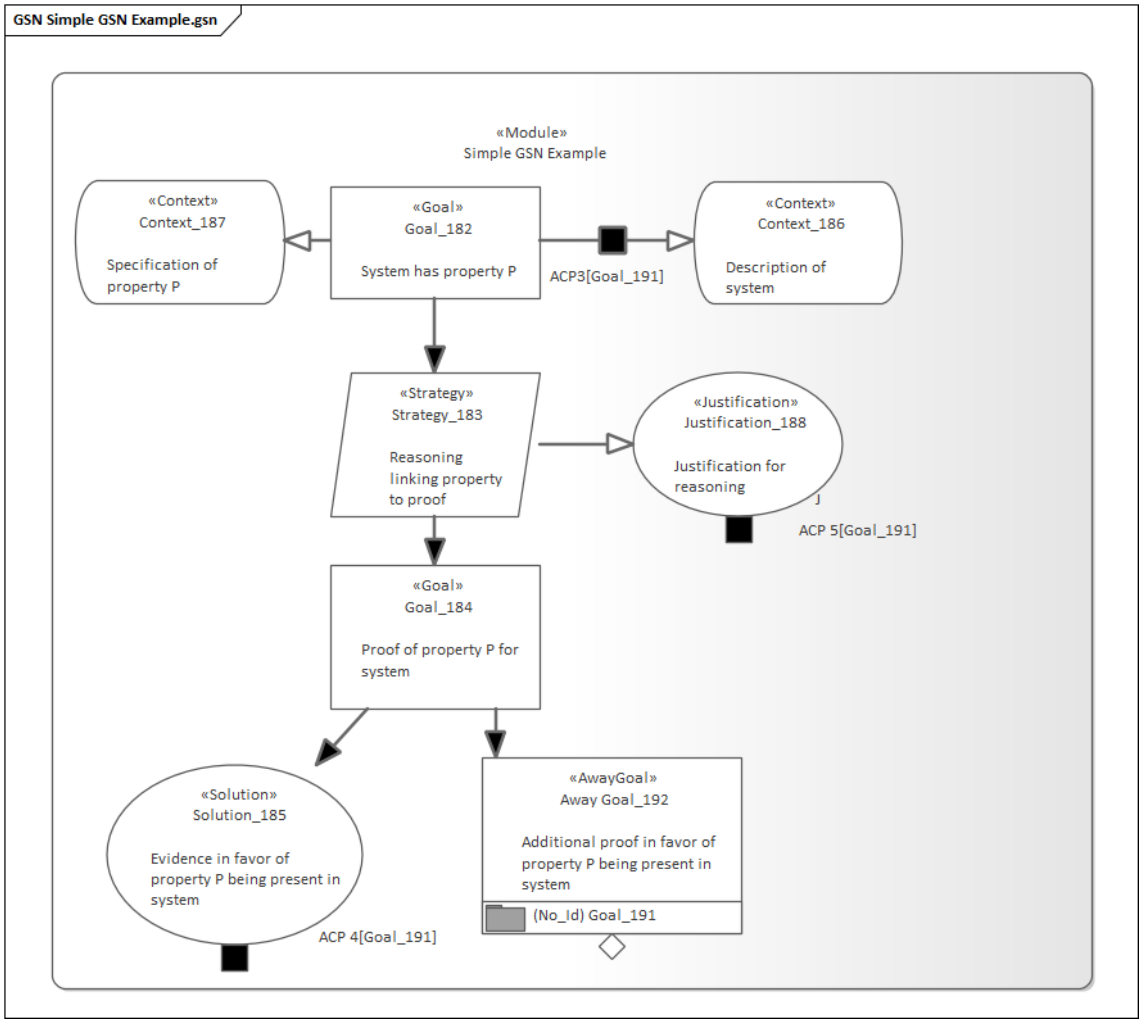


Figure 4 – Abstract ACP Example

Dialectic assurance case development is supported through the annotation of existing elements as being 'defeated', and "Defeated By" relationships. Elements that are considered to be defeated are elements whose arguing power has been undermined or eliminated due to the revelation of new information or lines of argument. The elements that cause the defeat can be linked to the existing argumentation via "Challenges" relationships. An abstract example of this extension can be seen in Figure 5, where CSn1 'defeats' goal G1.

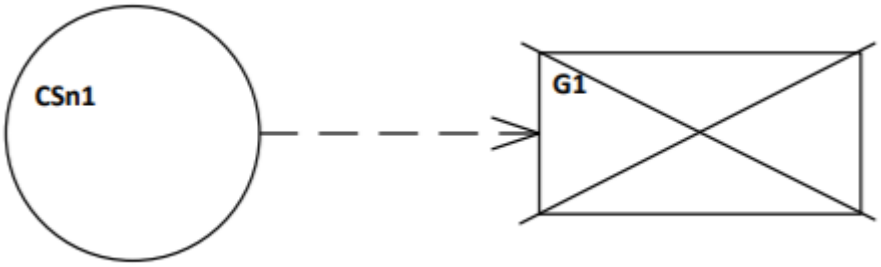


Figure 5 - Example of GSN Counter-Solution Defeating Goal (from [21])

2.5. Goal Structuring in Trust-based Digital Ecosystems

Within a digital ecosystem, systems and actors form coalitions for achieving common and individual goals. In a constant motion of collaborative and competitive forces and faced with the risk of malicious attacks, ecosystem participants require strong guarantees of their collaborators' trustworthiness. Evidence of trustworthy behaviour derived from runtime executions can provide these trust guarantees, given that clear definition and delimitation of trust concerns exist. Without them, a base for negotiating expectations, quantifying achievements and identifying strategical attacks cannot be established and attainment of strategic benefits relies solely on vulnerable collaborations.

For uplifting the assurance case from systems to the level of ecosystems we have examined the relationship between goals and trust and we've created a formalism for goal representation. We delimit the trust concerns with anti-goals. The anti-goals set the boundaries within which we structure the trust analysis and build up evidence for motivated attacks.

Engineering digital ecosystems around open adaptive systems has become the enabler of technological advancements. Systems and devices from different manufacturers and even from different application domains interact and collaborate to achieve higher level goals, which would not be possible without such comprehensive collaboration. Moreover, there is a trend towards more continuous engineering, i.e., organizations and their developers dynamically enhance existing systems with runtime software updates that are continuously monitored.

We anticipate a stronger uptake of the agent-based system paradigm. Correspondingly, in the automotive domain for example, there would be smart software agents deployed on vehicles, which could also be updated dynamically at runtime. These smart agents can at the entry point of a highway collaborate with other vehicles for forming platoons. When driving in a platoon, vehicles benefit from reduced fuel consumption due to reduced air friction.

However, the complex dynamics of collaborative and competitive forces existing in an ecosystem rise multiple trust concerns for all ecosystem participants. Especially when competitive forces are hidden within declared cooperation and lead to malicious attacks. At the lowest operational level, a vehicle accommodating a software update requires strong guarantees of trust from the smart software agent. Actors with declared collaborative goals that actually act in competition can insert malicious behaviour together with the software update. Being received as black boxes by the host vehicle, these updates can contain intentional malicious logic faults introduced with the scope of causing harm.

In the scenario provided above, the smart software agent can suddenly accelerate or decelerate and cause multiple car crashes within the platoon. Such a behaviour can be caused by logic bombs [2] that remain dormant in the host for a certain amount of time, and trigger when an event happens, or certain conditions are met.

Trust is an essential enabler for the emerging trend of digital ecosystems. Without trust, user acceptance and thus market success would be impacted or even prevented. Further, not only user trust is required, but also trust between companies and other stakeholders (e.g., legislators and official bodies). Both aspects translate into the requirement that systems in the field need to have a basis for computing trust be-tween themselves for enabling cooperative relationships to form dynamically between formerly unknown participants. But the creation of trust requires mechanisms for accounting entities to their actions, responses, achievements and failures in a way that also enables

negotiations, decision making and ultimate identification of undesired behaviours. In this sense, goals are concepts that enable analysis and modelling of stakeholders' interests and concerns [22]. A goal is evidence of an accepted objective fulfilled by system agents in [23]. In the area of safety in particular, system functions, regarded in our work as operational goals, have been formalized for enabling safety argumentation. The topmost priority of trust evaluation of systems operating in the field is their safety. Also, in the safety domain, a wide range of all possible deviations and formalization of operational goals have been defined. Therefore, at the operational level, it is enough to consider definition of anti-goals from safety as the one presented in [24]. For trust reasoning at higher levels, however, we adapt the goal formalization from the safety domain by considering a two-fold approach: identification of goal artifacts used in literature and analysis of directive documents, such as the ones from the European Commission.

Digital ecosystems until now have been engineered with considerations of separated trust concerns that have been focused on distinct areas such as robustness or user trust. But the hybrid and complex nature of ecosystems dynamics characterized by interactions among diverse actors such as users, businesses, official bodies, systems, system components and developers require a unified consideration of trust concerns. Ecosystems need an instrument for health self-regulation that can, for example support a trustworthy reaction of a developer to user demands through provision of on-the-fly software updates. Only through a self-regulating mechanism that enables continuous scrutiny of its health, an ecosystem can grow well. The health of an ecosystem is an indicator of how well the business performs [25]. In this work, we examine the relationship between goals and trust, and we introduce a formalism for goal representation. The formalism captures key aspects of goals, enables their expression in a natural language and tracing between multiple levels of computation. We consider goal evaluation to be the mechanism for self-regulating ecosystems, the one that can bring transparency in the trust building process and enable re-considerations of tactics and strategies. For this, we extend the previous platform for runtime prediction and trust computation [26] by considering the goals of ecosystem entities. In this way, evidence gained from runtime computations supports the tactical decisions of ecosystem entities and their strategic analysis, which in turn supports reconfiguration. Provided as an extension of a previous reference architecture for trust-based digital ecosystems we have introduced in [27], the current work continues with the demonstration of concepts expressiveness and reusability, by continuing with examples from the automotive and energy domains.

Dynamic Goals analysis

Goals can be viewed as the motivation between entities and their actions. They give a base for judging achievements and failures, as they enable negotiations and decision-making. When represented in a machine-readable format, they support the automatic reasoning of trust, through runtime computation of reputation. For enabling goal representation, we continue with formalizing their definitions in three layers: strategic, tactical, and operational. The strategic goals are given by high authorities, such as governments and associations of organizations. From the tactical to the operational level, we follow a top-down approach, in a 4C step-wise-refinement of goals: From Cooperation to Collaboration (tactical), Coordination and ultimate Communication (operational). We based our top-down argumentation and decomposition of goals on the work of Jones [28]. In this sense:

- Cooperation is the work on a task that shares the profits or benefits of doing so. It sets out a win-win benefit between two entities.

- Collaboration is the willingness of an actor to work jointly with another one on a given task. This can portray a mayor benefit for the entity requesting collaboration and a minor benefit for the collaborating entity.
- Coordination is the process of causing parts to function together in a proper order. There is no notion of benefit included here. At this level, systems, components, processes and tasks at most implement coordination mechanisms.
- Communication is the exchange of information and forms the basis for all the other upper C's concerns.

Starting from existing goal formalization practices used in the safety domain, such as the Goal Structure Notation (GSN) [7] [6] [29], we continue with a two-fold approach for formalizing goals for trust. We use the goal artifacts identified and mapping of goal artifacts identified in the literature and safety formalism to information present in directive documents that present strategic developments of industries in Europe. We've then deepened the analysis of the European strategic goals by surveying directions into two major domains to which the directive document is pinpointing: the automotive and energy domains.

Formalization of Goals

For defining strategic goals, we have looked at the highest strategic directives in Europe and we have surveyed the European Green Deal[3]. In this regard, the European Commission is an actor of a digital ecosystem that states strategic goals for organizations that take part in the ecosystem. For example, the European Commission states that for achieving the target for 2030 of reducing greenhouse gas emissions by at least 50% compared to 1990 levels, and no net emissions by 2050, it is essential for all sectors of economy to work towards a sustainable future. Policies needs to be revaluated for clean energy supply across the economy, industry, production and consumption, to name a few. One of the main strategic goals of the European Commission is to transform the European economy while creating a sustainable future.

For the strategic goal, we have identified five different artifacts, namely:

- The **Ecosystem Entity** is the non-cyber-physical part of the ecosystem, to which a strategic goal is applied. It is the one that supports the consequences and/or the benefits.
- **The Response** is the desired property that the ecosystem entity is planned to hold over time.
- **The Stimulus** is the condition that triggers the initiation of the strategic goal.
- **The Motivation** is the incentive for creating the response of the strategic goal. It is a trigger for adapting ecosystem entity own behaviour towards goal achievement.
- **The Quantified strategic benefit** is a quantitative achievement of a goal.

In this way, a strategic goal can be expressed using natural language in the following way:

"Ecosystem entity shall ***response*** when ***stimulus*** in the context of ***motivation*** with the benefit(s) ***quantified strategic benefit***."

Following the above structure, the following strategic goal has been defined based on the text in the "European Green Deal" document[4].

[3] https://ec.europa.eu/info/strategy/priorities-2019-2024/european-green-deal_en
[4] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52019DC0640

"All sectors of EU economy shall adopt European Green Deal when tackling climate and environmental challenges for creating a sustainable economy with the benefit(s) of achieving no net emissions of greenhouse gases by 2050".

For enabling the analysis of its fulfilment, we've further on decomposed strategic goals into domain strategic goals. One of these sectors of EU economy is the automotive domain which generates turnover of over 7% of EU GDP. One such group in the automotive domain is, ACEA (European Automobile Manufacturers Association)[5], a group of 16 major European automobile manufacturers, advocates of automobile industry. The association acts as a portal to provide expert knowledge on vehicle related regulation in the field of modern transportation. ACEA transforms strategic goals of governments into strategic goals of automotive companies. For example, ACEA provides action plans that supports the achievement of targets defined in European Green Deal with respect to mobility.

Tracing between strategic goals and domain strategic goals is achieved through decomposition of the *response* of the *strategic goal* into:

a) more concrete statements that become the motivation for individual domain specific strategic goals and
b) concrete responses of the entities that act in specific domains.

The template for defining domain strategic goals is:

"***Ecosystem entity*** shall ***response*** for ***motivation*** with the benefit(s) ***quantified strategic benefit***."

In the energy domain, we've defined the following domain strategic goals:

(Smart Grid) Domain Strategic Goal 1: European member states shall update national energy and climate plans by 2023 for contributing to EU-wide targets with the benefit(s) of reaching the 2030 climate ambition.

(Smart Grid) Domain Strategic Goal 2: The Trans-European Networks – Energy (TEN-E) Regulation shall foster the deployment of innovative technology and infrastructure for upgrading existing smart infrastructure with the benefit(s) of transitioning to clean energy at affordable price.

If at the strategic level, authorities define goals for the benefit of organizations, citizens and other ecosystem participants, at the tactical level, goals are defined for enabling system cooperation in the field. These are open declarations of an ecosystem participant, that other ecosystem participants can relate to.

For enabling tracing, the response of domain strategic goals is refined into tactical activities and the motivation into common benefits. For definition of tactical goals in the energy domain, we have surveyed the Smart Grids Task Force Expert Group 4 -– Infrastructure Development [30] that specifies the KPI (Key Performance Indicators) of cooperation within the energy sector and we have surveyed literature papers that describe tactics for cooperation, like the ones presented in [31] and [32].

(Smart Grid Tactical Goal 1): Distributed Energy Resources shall form coalitions when they can only provide fluctuant energy for satisfying the energy demands of users.

(Smart Grid Tactical Goal 2): Distributed Energy Resources shall transmit and distribute energy when they overproduce for reducing congestion risks in transmission networks.

The ultimate trust evaluation relies on runtime computations that create evidence of correct operation in the field. For achieving tactical goals, at the operational level,

[5] https://www.acea.be/about-acea/who-we-are

systems and system components respond to stimulus and operate in certain contexts. The stimulus of tactical goals become the context of operational goals. System functions that implement operational goals are activated in established context. The context of operational goals is a combination of internal and external states of the system.

We have defined the following template for natural language expression of operational goals.

- ***Ecosystem Entity*** is a system component, hardware resource or software component that implements a system function.
- ***Response*** is the output provided by the system function.
- ***Stimulus*** is the input provided to the system function.
- ***Context*** is the environmental part that starts the execution of a system function.

With the following template, goals can be defined at operational level:

"***Ecosystem Entity*** shall ***response*** when ***stimulus*** in context of ***context***".

In the smart grid domain, through the deployment of software smart agents, connectors boxes within a Distributed Energy Resource (DER) can autonomously form coalitions for satisfying the tactical goals such as provision of flexible amounts of energy. For this, at the operational level, the following goals need to be fulfilled:

(Smart Grid) Operational Goal 1: The connector box, part of a DER shall transmit state information when it receives a triggered request.

(Smart Grid) Operational Goal 2: A Virtual Power Plant shall start a broadcast for bids when it receives information about deficit of energy production.

The considerations outlined above (across the current section) can be used to inform the decomposition of assurance arguments, highlight areas of further investigation with respect to safety and security (e.g., where goals require further development or supporting evidence), and identify means of managing safety and security risks at the according strategic/tactical/operational level.

2.6. Conditional Safety Certificates (ConSerts)

Conditional Safety Certificates (ConSerts) [33] [34] capture modular, conditional, and pre-assured safety concepts, that enable dynamic reconfiguration of the underlying system based on observed changes of the operational context. As a reminder, safety concepts specify how a given system addresses safety risks e.g. by mitigating their effects.

However, when following the standard guidelines of e.g., ISO 26262, safety concepts must be specified during development, and depend on assumptions regarding the operational context. Due to the uncertainty regarding the exact conditions experienced during operation, typically worst-case assumptions are adopted. The implication is that the corresponding safety concepts typically restrict operation and/or system performance in order to maximize the likelihood of transitioning to a safe state of operation. This limitation is even more severe when considering adaptive systems, or systems operating in highly dynamic environments.

ConSerts aim to address this limitation, by enabling systems to adapt dynamically to changing conditions, while still assuring the safety of the adapted states of operation. To achieve this, safety concepts are modularized by viewing the functional architecture of their corresponding system as a set of service contracts between required and provided services of a Service-Oriented Architecture (SOA). Additionally, variability due to the system providing a service, the services it depends on, or its environment are used to gradate the guaranteed and demanded services in terms of functional and non-functional qualities.

An example of an abstract ConSert can be seen in Figure 6. In the figure, a ConSert of a hypothetical system providing a service X (as noted at the top of the figure) is depicted. The service has been specified in the form of a contract, providing a set of guarantees, which depend on the set of satisfied demands from services required by service X. In the case of this example, service X poses demands to service Y.

Specifically, service X can be offered at 3 levels of guaranteed quality (with the last being no guarantees) and can provide some level of guarantee if service Y can satisfy its corresponding demands. The guarantee and demand logic specified in the contract is depicted using Boolean logic gates, which in the case of the example is a logical ‘AND’ gate, meaning all supporting elements (linked at the bottom of the gate) must be logically ‘True’ for the gate to satisfy the linked guarantees.

In summary, service X can be provided with guaranteed level 1, if service Y satisfies a demand of level 1, and the system has confirmed from its own runtime evidence (RtE) that a predicate is valid. The logic of the predicate is specified externally to the ConSert. A similar situation holds when service X is provided at guaranteed level 2, depending on service Y being provided at guaranteed level 2 and the RtE predicate being satisfied. If neither level of guarantee can be provided, the service can still be provided (in this case) without guarantees. As the RtE and the demands provided by Y are updated at runtime, the guarantees of X are updated accordingly.

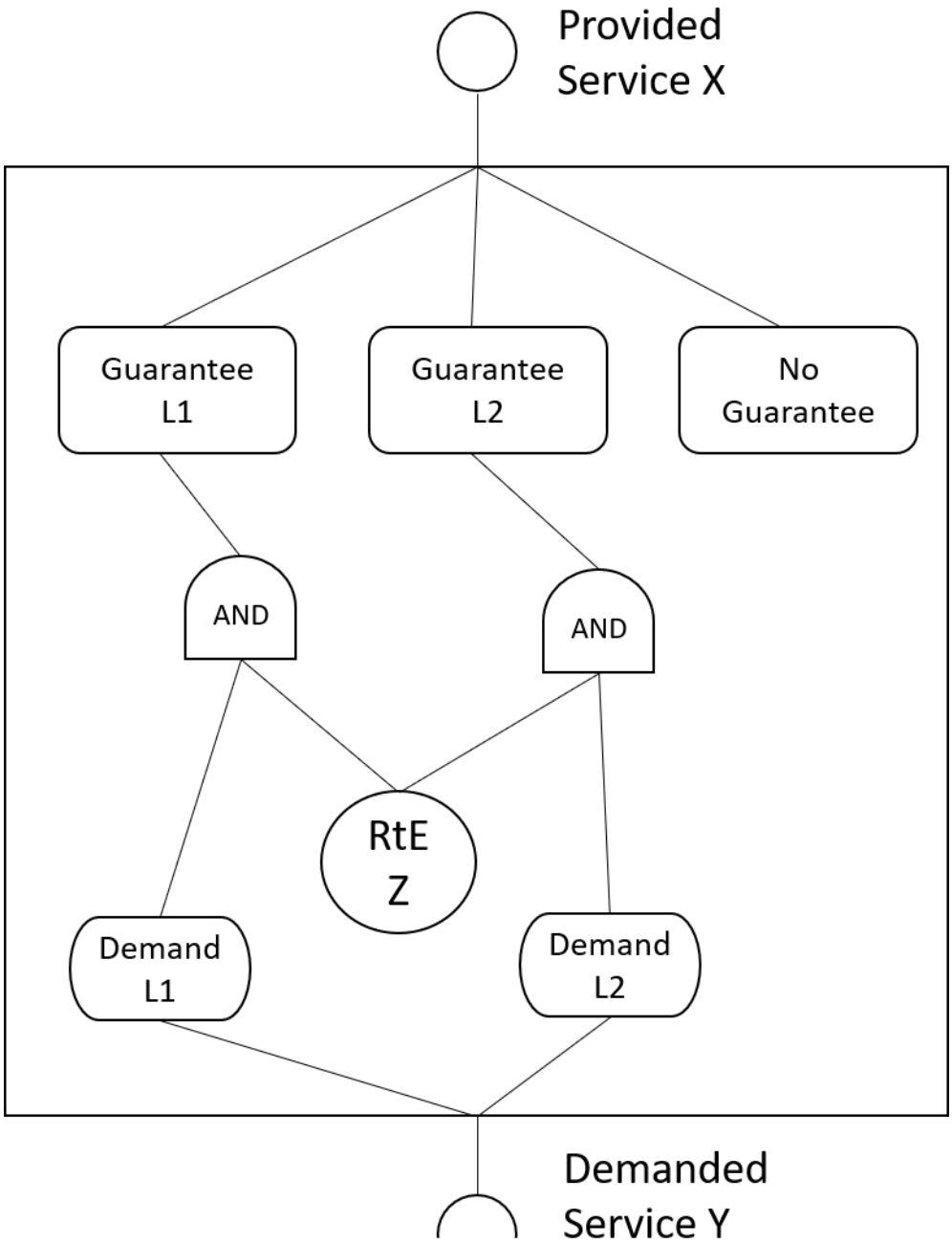


Figure 6 - Example of Abstract ConSert

2.7. Digital Dependability Identities (DDIs)

In the context of dependability-critical systems development, the paradigm of model-based engineering has been widely acknowledged [35] [36] [37] [38], promising a central model around which nominal development and dependability-related development processes can be synchronized and performed more efficiently, especially when supported by appropriate software tools. However, due to the complex and varying applications, a plethora of highly diverse domains, models, methods, and tools are available, making interoperability across organizations or even across teams within organizations challenging.

A particular challenge can be found in the coordination of safety and security assurance processes, as these topics are often highly correlated in systems integrating both electronic and physical components, as well as communication infrastructure over potentially open channels. For the model-based paradigm to be effective, both the safety and the security concerns must be captured and expressed using terms that, if not common, are at least accurately translatable across both safety/dependability and security engineering teams, such that they can collaborate and exchange information and results efficiently.

Digital Dependability Identities (DDIs) [39] [34] [40] [41] are modular, composable, and executable models of dependability regarding a specific system (or system-of-systems). During development, DDIs can act as exchangeable models that compose partial dependability analyses of the underlying system into a complete dependability assurance case. Such DDIs can be iteratively constructed as information from development processes becomes available, ideally by extracting directly from results of associated tools. Therefore, DDIs can serve as appropriate intermediate models for managing the complexity of the information exchanged during dependability-related development processes. Additionally, they allow capturing and translating concerns across domains, (e.g. safety and security) consistently, thereby alleviating communication across corresponding processes, teams, and tools.

2.8. Intrusion Tolerance Countermeasures

Borrowing from the well-known Fault Tolerance principles, Intrusion Tolerance aims to guarantee that a system works correctly even when some of its parts are compromised. Similar to Fault Tolerance, Intrusion Tolerance consists of several techniques aimed at counteracting an error from turning into a failure. The specific emphasis on *intrusion* instead of the generic *fault* refers to the malicious characterization of an intruder launching an attack, exploiting at best the acquired knowledge. Here, we refer to the more sophisticated chain illustrated in Figure 7 than the classical fault-error-failure in [2]. As illustrated in the figure, an attacker exploits a system/component vulnerability to launch an attack that, if successful, leads to an intrusion, seen as an internal fault. This fault may generate one or more errors that, if not properly managed, can lead to the failure of the service provided by the system/component. This is in line with the AVF fault model in [42] [43].

To avoid/mitigate the potential failures, Intrusion Prevention [44] and Tolerance are put in place to cope with the attacks and their consequences. Prevention is the first defense against intrusions, but since prevention cannot be assured to be totally effective, intrusion tolerance is also needed. In Figure 7, small "holes" within the area illustrating a defense technique represent weaknesses of the technique itself in accomplishing its task. Attack removal and vulnerability removal are radical measures to cope with the source of the intrusion, so avoiding that similar attacks can be perpetrated again.

Intrusion tolerance techniques primarily include intrusion detection [44] [45], intrusion removal [46] and masking of intrusion-induced errors [43].

In this deliverable, the focus is on the last category of Intrusion Tolerance means, where redundancy of components is exploited a first line of defence while the other techniques collect enough knowledge to deal more radically with the presence of the intrusion. Moreover, here the emphasis is on a single component that is selected for higher protection through redundancy-based IT, where replicated components (variants) do not interact with each other, but only provide their output to the component in charge of collecting and manipulating variants results. Hence, tolerance paradigms applicable in contexts where multiple interacting components are in place are out of scope. For example, solutions resilient to byzantine faults in distributed systems have been deeply investigated, where the ability of the redundant components to interact with each other is exploited for tolerance purposes (see [47] for a survey). Another example is secure multi-party computation, a sub-field of cryptography with the goal of creating methods for parties to jointly compute a function over their inputs while keeping those inputs private (e.g., [48]). Again, interaction among involved components is the differentiating aspect with respect to the approach pursued in the current work.

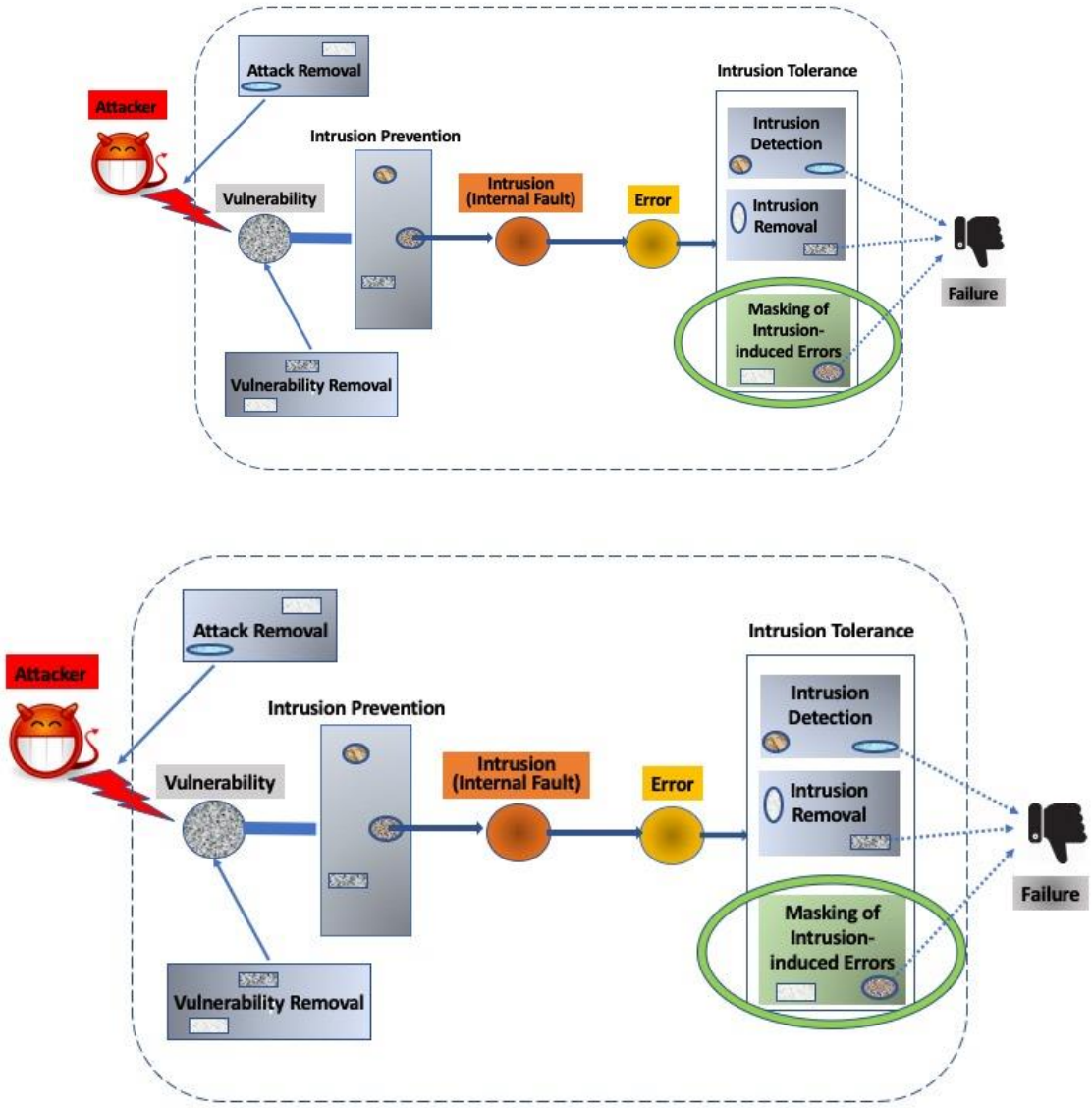


Figure 7 - Representation of the Attack-Vulnerability-Intrusion-Error-Failure chain, and categories of techniques to cope with it

Based on the long-dated experience with fault tolerance, and considering the peculiarities of an intentional attack, the proposed general conceptual framework for Intrusion Tolerance develops along four major dimensions, as shown in Figure 8.

Figure 8 - The proposed conceptual framework for redundancy-based Intrusion Tolerance

These dimensions can be considered the *ingredients* to work with in the design of redundancy-based Intrusion Tolerance mechanisms when facing a specific application context.

2.8.1. Attack model

The attack model indicates how the cyber-attack is characterized. From the intrusion tolerance perspective, essentially what is relevant is to know the consequences of the attack, more than the dynamics on which vulnerabilities it exploits and the path to the successful intrusion. These last are vital information to accomplish the attack detection and treatment, but less relevant to carry out masking of the generated intrusion. Attack consequences are then connected with the security attribute that is primarily impacted, namely the well-known triad: *confidentiality*, *integrity* and *availability*. As a brief recall from [2]: i) confidentiality is preserved in absence of unauthorized disclosure of information; ii) integrity is preserved in absence of improper system alteration; and iii) availability is preserved with readiness for correct service. Similar to [49], the manifestation of a successful attack on the compromised components can be summarized as follows:

- *Functionality Change*, which is the delivered results are incorrect. This means that the compromised components experience a failure in the value domain. The impact is mainly on the integrity property.
- *Performance Degradation*, which is the results are delivered late or, in the extreme case, they are omitted. This means that the compromised components experience a usage failure at a specific moment. The impact is mainly on the availability property.
- *Information Leakage/Improperly Accessed*, that is sensitive information are revealed. The impact is mainly on the confidentiality property

Intrusion tolerance techniques, by masking the presence of compromised components, are mainly directed to preserve integrity and availability. Regarding confidentiality, such techniques are not effective; attack prevention is the reference approach for this property. Therefore, confidentiality is left out from the conceptual intrusion tolerance framework under development in this work.

2.8.2. Categories of system components targeted by attacks

Cyber-attacks can be launched to all the components of an ICT system, and typically an attack develops through several of them to be successful and lead to an intrusion. From the attacker perspective, the ICT components can be considered as belonging to three major categories:

- *Computing element*, which is a component that is devoted to performing some kind of functionality, to provide a service to the requesting entity (a user or another component). Operating systems primitives, software applications and enterprise software are typical examples of this category.
- *Communication elemen*t, which is the means through which information is delivered to/from computing elements, users and storage. The internet and the several wireless networks technologies are typical examples of this category.
- *Data storage element*, which includes different storage technologies used to retain digital data within a computer system architecture. The term storage may refer both to a user's data generally and, more specifically, to the integrated hardware and software systems used to capture, manage and prioritize the data. This includes information in applications, databases, data warehouses, archiving, backup appliances and cloud storage.

2.8.3. System model and failure assumptions

Without going into details, such system models range from monolithic structures to distributed interacting components of different granularities, including SOA (Service Oriented Architecture) [50], microservice and SoS (Systems-of-Systems) paradigms [51]. A discussion on pros and cons with each system model is out of the scope of this study. What is relevant to notice is that the architectural solution is typically chosen on the basis of the functional and non-functional requirements, as well as cost implications. Since flexibility and scalability are among the most relevant requirements to drive the selection of the system architecture, the current trend is to increasingly evolve from the monolithic structure to forms of distributed computation. However, in addition to other considerations, it needs to be mentioned that there are long-lived systems, originally developed as a monolithic architecture, which cannot undergo significant redesign, but need to be enhanced from the resilience perspective. Therefore, the interest in monolithic-based solution appears to be still significant.

Another important aspect associated to the structure and operation of an ICT system is the assumed *failure model* for the system components (due to accidental faults and/or intentional attacks). In line with the considerations on the effects of attacks, experienced failures can be in the *value domain* (an incorrect value is delivered/transmitted/stored) or in the *time domain* (a value that violates the time constraints is delivered/transmitted).

2.8.4. Type of redundancy

Another relevant dimension is the type of redundancy to adopt, that is the characterization of the forms of redundancy that can be put in place. When the redundancy is obtained employing just replicas, i.e. identical copies, if the attacker is successful in compromising one replica, it is expected that the intrusion is immediately successful in all the other replicas, due to the common vulnerabilities. Therefore, diversity is advocated, as a basic and powerful instrument to mitigate intrusion propagation. In practice, instead of replicas, the redundant structure adopts variants, which are functionally equivalent components developed with some form of diversity.

As shown in Figure 9, common mode vulnerabilities when designing, implementing and deploying redundancy to protect a system component can have different origins. In the figure, three major sources are identified:

- *functional influences*, which has primarily to do with the same design vulnerabilities present in the redundant components, or the adoption of a common execution environment exposing the same vulnerabilities. An intruder can exploit these vulnerabilities so that the affected components provide the same incorrect output, or the same late/unresponsive behaviour. A variety of means to cope with functional influence have been proposed in the literature (e.g., in [52]), some of which are reported in Figure 10 (e.g., different development teams, different programming languages, compilers, run-time supports, etc.) As will be discussed when presenting the redundant schemes, the usage of diverse components adds some difficulties to the definition of the redundant structure, namely the need to account for correct outputs resulting in non-coincident values.
- *locational influences*, which exposes all the redundant components located in the same physical or logical partition of a system to be isolated by an attacker (e.g., through intrusion in the communication network), without the possibility to receive inputs and to provide outputs to the adjudicator unit. As shown in Figure 10, locational influence mainly consists of adopting redundant (diverse)

communication channels, and/or distribution of the redundant components in different physical/logical sites.

- *administrative influences*, resulting in potential massive intrusions by exploiting social engineering, when the redundant components are subject to the same security management policies. Diversity measures to cope with such problem include the adoption of different security management policies for the different redundant components, and possibly different administrative domains, as shown in Figure 10. Of course, some form of coordination among such different policies/administration domains turns out to be needed.

Based on the conceptual framework depicted in Figure 8, in Section 3 classical redundancy-based fault tolerant approaches are revisited from the security perspective.

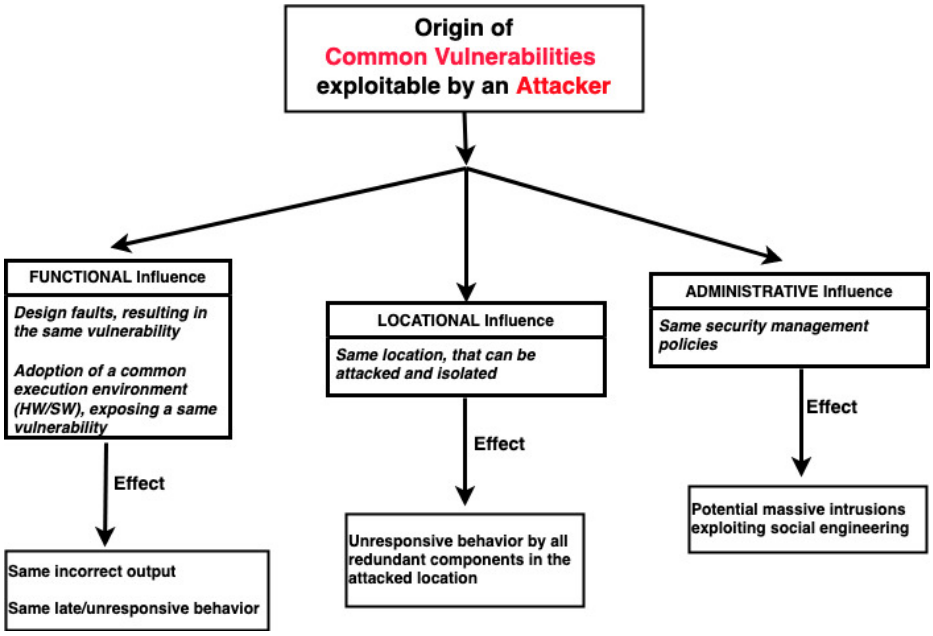


Figure 9 - Origin of common mode vulnerabilities, exploitable by an attacker

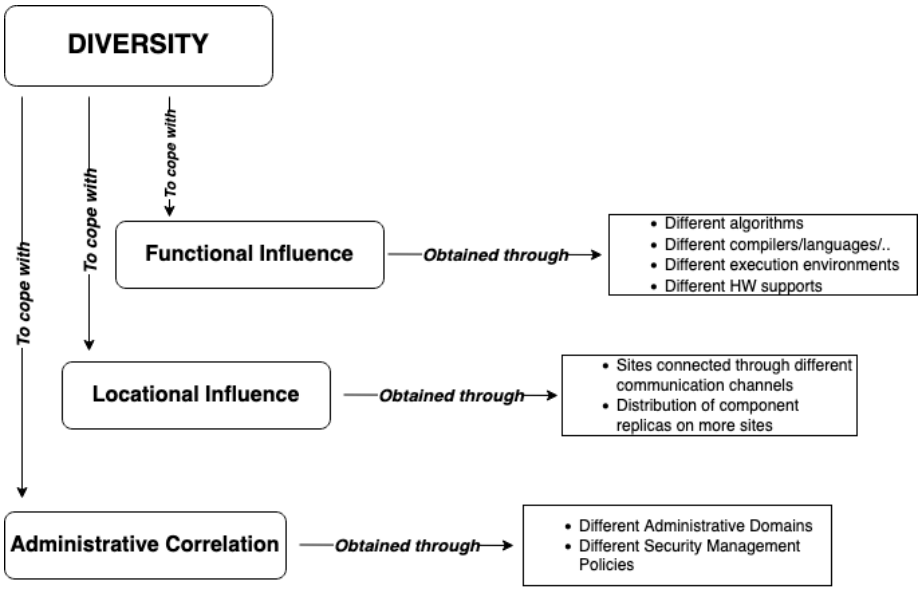


Figure 10 - Diversity approaches to cope with correlation types

3. Redundancy-Based Intrusion Tolerance Countermeasures for Design-Time Risk Mitigation

Slightly extending the taxonomy in [53] [54], diversity-based fault tolerance scheme to mask the presence of errors is characterized by four major design issues: i) decision on the measures to adopt for enforcing diversity in the redundant structure under development; ii) selection of variants to employ in the redundant configuration (e.g., how many variants to employ, each developed according to which diverse methodology, thus showing a required reliability level); iii) decision on the execution pattern of the selected versions; iv) decision on the adjudicator function to adopt for selecting the only output from the set provided by the employed variants. The designer mainly addresses these issues based on specific needs of the application under development (including dependability requirements, as well as other requirements in the time domain and operational context), available development environment facilities, and reference fault tolerance architectures.

Approaches to obtain *diverse* functionally equivalent versions of the component that needs redundancy have been already addressed in Section 2.8.

Regarding the execution model for the variants, both *sequential* and *parallel* execution are implemented in the reference schemes. When adopting sequential execution, as typically done in the Recovery Block approach, a reliable checkpointing mechanism is needed, to save the state of the system before any variant starts executing and from which an alternate variant starts its execution, should the previous variant fail. Parallel execution implies concurrent execution of the variants, thus requiring adequate computer resources and the use of mechanisms to assure that the same input is provided to all the variants.

The *adjudicator* component plays a very critical role in the overall redundant organization, being the entity that takes the final decision on the outcome of the redundant computation. A variety of adjudication functions have been proposed in the literature. They belong to two major categories: *Voters* and *Acceptance Tests*.

Voters make a judgement on the set of variants results and are typically employed when the variants follow the parallel execution pattern. Several kinds of voters have been proposed to select the outcome from the set of results provided by the variants, such as majority voter, consensus voter, median voters [53] [54]. Note that the presence of diversity requires more sophisticated voting solutions than simple bitwise comparison (correct variant results are expected to be not perfectly coincident).

Acceptance tests make an absolute judgement on each single-variant result. This kind of adjudicator is typically used when the execution of variants is sequential. Popular acceptance tests are based on satisfaction of requirements or reasonableness test (see [54] for more examples). Hybrid adjudicator forms that employ combinations of voter and acceptance test have been also explored, such as in [55]. A final observation on the discussed adjudicator categories is that voters are universally applicable, since they are based on syntactic comparison, while acceptance tests have to do with the semantic of the function performed by the variant, and a sufficiently accurate test of its result can be difficult or impossible to define. In [56], an optimal adjudicator function was proposed, that has the highest theoretically possible probability of producing a correct result for any input to a particular redundant component. It exploits probabilistic knowledge about errors/faults in the subcomponents of the fault tolerant component. Although not exploitable as a practical adjudicator, the concept of optimal adjudication is useful both as an upper bound on the probability of correct adjudged output obtainable and as a guide for design decisions.

To help addressing omission failures experienced by variants, the adjudicator component is typically equipped with a *timeout* mechanism that terminates the waiting on a variant's result, after a predefined time interval determined on the basis of maximum execution time for the variant under consideration. In the following, it is assumed that each addressed intrusion tolerance scheme adopts a timeout for this purpose.

At first sight, it could seem unnecessary to tailor n Version Programming [57] (more general mr), Recovery Blocks [58] [55], n self checking, etc, to the security context because, at least from [2] on, it is clear that fault-tolerance already addresses intentional faults (namely attacks). Nevertheless, in Section 2.8 it has been discussed how the peculiarity of intrusions requires the employment of forms of diversity to better protect the system/component under development. In classical redundancy-based fault tolerance schemes, diversity is advocated to cope with design faults resulting in common mode failures, but its need in addressing accidental faults is less stringent/radical than in case of intrusions (among the diversity approaches depicted in Figure 10, measures addressing functional influences are mainly considered).

The stronger role played by diversity is a first distinctive aspect of redundancy-based intrusion tolerance schemes with respect to corresponding fault tolerance ones. To further enhance the efficacy of offered Intrusion Tolerance solutions, a few other features that have been proposed to support fault tolerance and/or security properties in general are exploited.

3.1. Additional protection measures

As additional protection measures to enhance the efficacy of Intrusion Tolerance schemes, the concepts of *rejuvenation*, *locality*, *access control* and *confusion* have been selected as promising candidates. A brief recall of each of them, with considerations on their employment in the proposed IT schemes, is in the following.

3.1.1. Locality

Location diversity, consisting of placing several physical components of a system in different sites, is recognized since long time as a good practice to cope with physical threats, like natural disasters (e.g., floods or fires) or basic service outages (e.g., electrical outage). When deliberate attacks are considered, as in intrusion tolerance, this measure becomes even more relevant. Interestingly, location diversity can be easily joined with diverse administration domains characterizing the different sites, so further improving the defense against attackers [59]. In the following it is assumed that the defender has s sites at disposal and can distribute the variants among the sites. Utility functions (e.g., input scatter, output gather, acceptance tests, adjudicator, etc) are deployed on a special site not counted.

Notice, though, that scattering data and code among on-premises and/or commercial data centre to improve on system resilience, has the potential drawback to degrade confidentiality, depending on the accessibility conditions to the chosen diverse sites, as discussed in [60] for embedded systems and in [61] in the context of Byzantine Fault Tolerance. Thus, side effects of location diversity have to be carefully analysed and managed. Being the subject too application specific, it is not addressed here.

3.1.2. Rejuvenation

For long-living systems, rejuvenation [62] enhances fault-tolerance: once in a while, each replica is subject to some form of clearance/rejuvenation in order to reduce its failure rate or the frequency of intermittent faults, so that the entire fault-tolerance scheme is improved.

While relevant to contrast any kind of malfunctions producing erroneous behaviours of a system/component that tend to increase along time, the benefit of rejuvenation is essential in the context of intrusion-tolerance, where an intelligent attacker may have enough time to successfully accomplish its intrusion. In fact, if clearance actions are effective enough, rejuvenation reduces the time window for an attack to be successful to be just the time between two consecutive clearances. To this purpose, rejuvenation should:

- take place with high frequency, so that the attacker has to act as quickly as possible, potentially making mistakes that trigger intrusion and information leak detectors that are in place,
- be coupled with diversity, i.e., each rejuvenation introduces as many changes as possible for reducing the correlations between pre- and post- clearance, ideally generating a completely new variant.

In [63] the authors make a distinction between proactive and reactive rejuvenation policies in the context of distributed systems, where the interaction among components allows some form of reciprocal *diagnosis* based on perceived behaviours and consequent triggering of a rejuvenation phase in case there is the suspicion of a malfunctioning component. Instead, when focusing on individual component replication as addressed in this work, rejuvenation can be mainly applied as a proactive defensive measure. However, to account for potential self-checking features a variant of the redundant scheme could be equipped with, the rejuvenation can be either scheduled at

predefined time intervals or activated when some critical event is perceived (e.g., the variant itself could apply internal checks to reveal suspicious behaviour). Of course, rejuvenation is a costly operation and resorting to it with high frequency can become too onerous, especially when applied for contrasting attacks. In fact, the rejuvenated variant is expected to be significantly diverse from the original one; acting at functional level should assure higher degree of diversity, although simpler *automatic diversity* forms (like change in name or position of files in the filesystem) or *obfuscation* techniques (e.g., compile the program in sophisticated, and always different, ways that make reverse engineering difficult) can be considered as well. In general, a trade-off needs to be carefully analysed between several involved aspects to find the suitable rejuvenation strategy (mainly in terms of rejuvenation frequency, degree of diversity for the rejuvenating variant, time needed to accomplish the rejuvenation, desired level of benefit from rejuvenation).

Of course, since rejuvenation does not guarantee full independence between pre and post versions of the variant from the attacker perspective, it cannot be the only defence mechanism in place.

In this deliverable, r indicates the maximum number of variants per site that can be under rejuvenation at each instant of time. The rejuvenation procedure requires an interval of time to be completed; therefore, when under rejuvenation, a variant skips one or more executions performed by the redundancy scheme it is involved in, until the rejuvenation phase completes.

3.1.3. Techniques to assure protection levels

Access control policies are typically applied to selectively restrict access to resources that play different roles, and a variety of access models have been developed to grant or reject an access request.

Access control in computer security has been widely investigated (e.g., [64] with reference to IoT technology). Through authentication and authorization, access control policies make sure users are who they say they are and that they have appropriate access to the intended resource. According to the criticality of a component, more or less stringent rules are applied to grant the access.

The redundancy-based solutions for Intrusion Tolerance that will be detailed in the following include components of different criticality: the functionally equivalent variants show lower criticality than adjudicator components responsible for selecting the outcome from the variants' outputs. From this we derive that the adjudicator component needs higher protection level than individual variants, in terms of reducing the ability to an intruder to access it as a resource to compromise. So, the need of adequate protection mechanisms and access control techniques is even more exacerbated in intrusion tolerance context, to avoid defeating the effort of costly redundancy.

For embedded or IoT systems, it is common to exploit a Root-Of-Trust to enhance security, and also fault-tolerance architectures can be complemented with such a mechanism [49]. Other solutions, such as resorting to a distributed adjudicator component to avoid the single point of intrusion are possible.

For our purposes, two layers of protection will be considered, indicated as L_0, L_1, where L_0 is the most stringent one. Of course, more a fine-grained solution with higher number of protection levels could be of interest in specific contexts/application domains. In more general terms, deciding which part of the intrusion-tolerant architecture has to be assigned a given layer is crucial, having profound consequences not only on implementation choices but also on the attack model, and then on the analyses the defender performs to oppose the strongest defence to potential attackers.

In the literature, there are proposals in several directions when addressing intrusion tolerance in specific contexts, such as:

- the adjudicator together with variants-adjudicator communication channels are in L_0, whereas the variants are in L_1 [49];
- adjudicator and variants are in L_0 but the communication channels are in L_1 [60];
- if the adjudicator is a simple voter, then its logic can be distributed onto the variants located in L_1, and only the interface with the output can be placed in L_0 [65].

Clearly none of the above is always better than the others, it depends on the context and the available resources. For instance, having most of the architecture in L_0 and only the communication channels in L_1 can appear from one hand too expensive, and on the other hand insecure because for an attacker it is easier to address the communications than the logic because this way almost no domain specific knowledge is required. However, this may not be the case because in some contexts (e.g., cyber-physical systems) both variants' and adjudicators' logic can be simple enough to be implemented in microcontrollers that are relatively cheap and easy to protect, or heavy and complex but implemented in containers that run on machines physically located in secure places, and communication channels can in turn be made intrusion-tolerant to reduce exposition to attacks.

3.1.4. Confusion

An accidental fault just happens. Conversely, an intentional fault (an attack) is the result of rational choices made by one or more adversaries, and usually strikes the variant that the attackers hypothesize to be the weakest ones. Thus, in intrusion-tolerant systems it is common to find *confusion* strategies aimed at decreasing the confidence the attackers have in their decisions or increase the attack cost. Available strategies have been developed for different mitigation purposes and so show different degrees of effectiveness. For instance, replacing some variants with *camouflage* ones, i.e., components that perform no operations but mimic the interactions that operating variants have with the environment [66], can add a sufficient level of confusion only if the attackers have limited resources, in particular of time. For a cyber-physical infrastructure, such as a Smart Grid, where the attackers can study the system and plan the intrusions for years, and where it is expected that foreign adversaries are willing to invest huge resources in the attack, camouflages are less effective. In the referred context, camouflages are of great help to set up honeypots aimed at gaining information about the attacks or to do detection, but to tolerate intrusions the most effective choice is to use extra but working variants and configure the tolerance scheme such that the adjudicator component decides (probabilistically or deterministically) which results to consider among the set of received ones. Of course, this solution is not always applicable due to its high cost, but is beneficial as much as for the analogous in distributed computing [67] [61].

Moreover, they are proposed as conceptual schemes, without any direct connection to specific application domains, each one typically characterized by consolidated design practices and agreed standards. Therefore, whether and which protection mechanisms suggested above are appropriate to be employed in a specific system design certainly depend on their consistency with recommendations dictated in the referred application domain.

To conclude this overview of security enhancing features, a graphical vision of the above concepts employed in redundancy-based fault intrusion is provided in Figure 11, where a configuration example is illustrated. It involves: 9 variants, of which 6 are predefined to be those whose outputs are considered by the adjudicator (named as *participating*

variants and labelled as pv_i), and the remaining 3 are ignored by the adjudicator (named as *non-participating variants* and labelled as npv_i), and 2 adjudicator components (labelled as A_i). A protection level is associated to each component (labelled as L_i) and distributed in four different sites (labelled as s_i). Periodic rejuvenation phases are also indicated. Note that, for visualisation purpose, this example has to be considered as a snapshot of the scheme configuration taken at a certain moment of its execution.

Finally, in Sections 3.3, 3.4 and 3.5 several redundancy-based intrusion tolerance schemes are described. It is clarified that such proposed schemes are not meant to be an exhaustive intrusion tolerance set; rather, they are examples of how the features discussed in this section can be exploited to adapt the traditional fault tolerance organization to cope with intentional attacks. So, there is openness to other interesting alternatives.

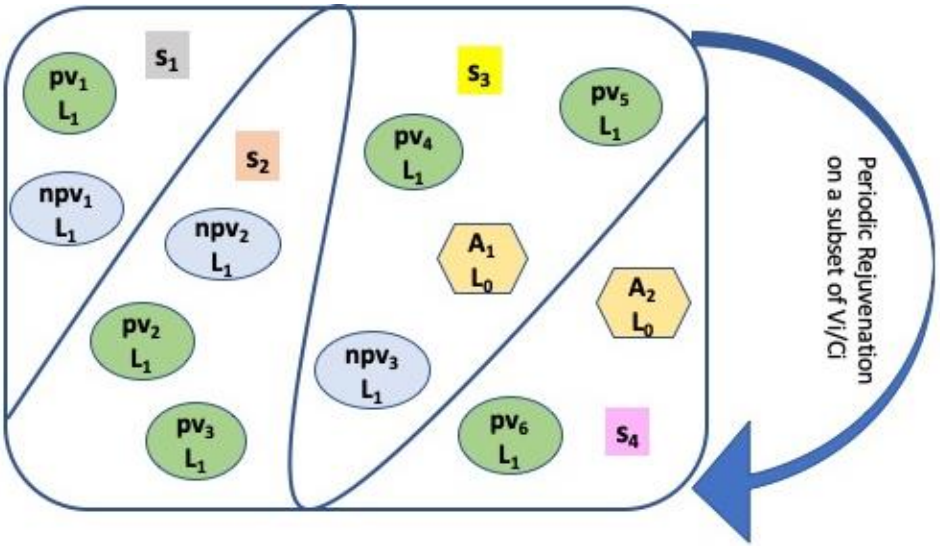


Figure 11 - Snapshot of a redundant-based intrusion tolerance configuration encompassing 6 participating variants (pvi), 3 non-participating variants (npvi), 2 adjudicators (Ai), with indication of their protection level (Li) and site where they are located (si).

3.2. Attack model

The assumptions on the attack model are detailed in the following. The first statement is that only cyberattacks are considered. Therefore, an ICT component, even when composed of a physical device and software managing/controlling its operational life, can be compromised only through the software part. An attacker has ability to:

- intrude the variants and
  - alter their result (value failure). The best strategy for the attacker is to try to compromise as much variants as possible, making them deliver the same (wrong) result, thus inducing a common-mode failure. f indicates the number of value failures (possibly of kind common-mode) generated during an execution of an intrusion tolerant scheme.
  - make their result unavailable, that is the compromised variants experience an omission failure. k indicates number of omission failures generated during an execution of an intrusion tolerant scheme.
- isolate 1 site among the s where the variants are deployed. The effect is that the results of all the variants located on that site became unavailable, and the adjudication function perceives an omission failure from these variants. The assumption of no more than 1 site under potential isolation by an attacker is in line with the works in [67] [61], and is made here for the sake of simplifying the presentation, but can be relaxed without invalidating the following developments.

The considered intrusion tolerant architectures can tolerate f arbitrary value failures (common-mode value failures, in the worst case) and k omission failures (comprehensive of both those intentionally caused by the attacker and those due to accidental causes).

The protection layer L_0 is assumed to be unattackable, so those functionalities put under this protection layer (namely, adjudication functions and possibly some of the variants) do not experience successful cyberattacks. However, variants subject to higher protection from cyberattacks can be still affected by accidental faults. Instead, for what concern the adjudication functions, given their higher simplicity and reliability, their failure (both in selecting a wrong result and in not recognizing the exiting of a correct result) is not directly accounted for in the proposed redundant architectures. Of course, their reliability needs to be considered when assessing the ability of the scheme to satisfy desired dependability properties.

3.3. Family of NVP-like Architectural Proposals

This section is dedicated to the family of redundancy architectures that follow the N-Version Programming (NVP) organization. After a brief description of the classical NVP fault tolerant architecture [57], a few solutions obtained from its adaptation in the context of intrusion tolerance are discussed.

3.3.1. The reference N Version Programming

As depicted in Figure 12, the N Version Programming (NVP)[6] comprises an adjudicator that receives all the results from the variants (or, after a timeout, works with those that are available) and, in the original formulation in [57], checks if there is a majority among the results. The variants are usually executed in parallel, although sequential execution has been investigated (the corresponding architecture is often called nVS) in contexts where computational resources are limited. If there is a majority, then this is the elected result that is sent in output. Otherwise, depending on the failure model, the component can switch to a benign failure state (e.g., in the context of Safety) or send in output a default value or choose one of the results exploiting other kinds of information, such as past knowledge about recurring errors (e.g., in the context of Reliability). This kind of adjudicator is called *simple voter*.

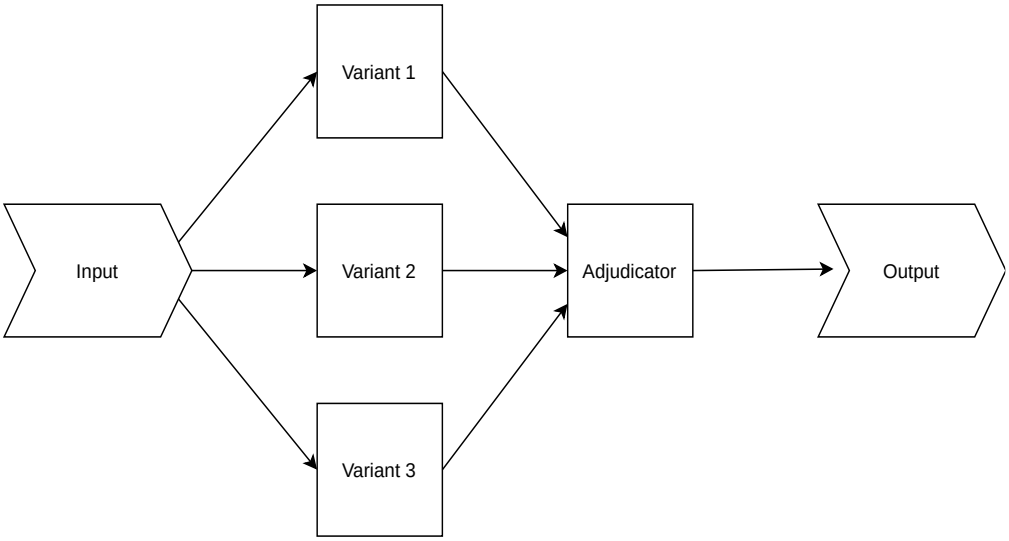


Figure 12 - Basic configuration of N Version Programming (NVP) employing n=3 variants.

Other kinds of adjudicators, such as variants of the simple majority (as presented in [54]), or exploiting more complex *syndromes*, such as additional information of the reliability of the variants (as for the optimal adjudicator in [56]) have been adopted. Actually, sophisticated adjudication functions can be defined, exploiting available information about the variants under execution to help selecting the (assumed to be) correct result (including previous disagreement with selected output, time to last rejuvenation/recovery) While keeping the voter simple in its logic certainly favours correctness of this critical component with respect to unintentional design faults, exploiting additional information at cost of introducing higher complexity appears appropriate in the context of Intrusion Tolerance, since it enhances the defender ability to perform rational choices.

[6] The term "n Version Programming" is not used in the software context only, but is commonly adopted in mixed software and hardware contexts. Notice that for pure hardware, the terms "n Modular Redundancy" or "static redundancy" are often preferred.

The NVP configuration appropriate in a certain application context, in terms of number of variants to employ and the adjudicator component to adopt, depends on the failure model that is intended to be tolerated. For instance, if the goal is to tolerate m failures (both benign and catastrophic), a suitable configuration requires $n = 2m + 1$ variants with the simple voter as adjudicator.

Many specific NVP-like redundant schema have been proposed in the literature, as those reported in the survey [53], which focuses on fault tolerant Service-Oriented Architectures. Efficient organizations have been also pursued, for instance by performing results comparisons as soon as they arrive, so to wait for only m to agree, thus improving on time performance, particularly relevant in real-time systems.

Of course, this scheme is fully recursive, meaning that a variant can in turn be implemented following the NVP schema.

Starting from the classic NVP fault tolerant organization just recalled, a few adaptations to the fault intrusion context are proposed in the following. A first, immediate NVP-based intrusion tolerance alternative, referred as basic iNVP and depicted in Figure 13, can be simply obtained by introducing protection mechanisms at level of communication channels (mainly to detect side-channel information leak) and encapsulating the voter component within a secure module (root-of-trust), as proposed in [49]. While this is certainly viable, in the following we concentrate on more sophisticated solutions, and therefore expected to be more resilient and/or less expensive (although this kind of quantitative assessment of resilience vs cost metrics is out of the scope of this work).

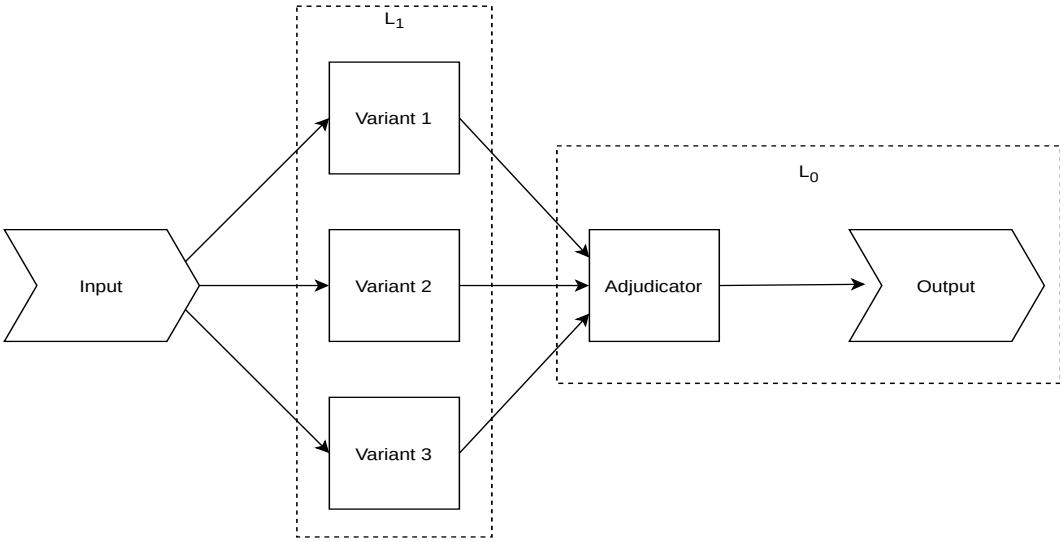


Figure 13 - Intrusion N Version Pprogramming (iNVP), in the example with n=3 variants.

3.3.2. Random Participation

Random Participation of h among n variants (iNVP-R) resembles the NVP structure and logical operation, but introduces a form of confusion to make the attacker's life harder. More variants than those strictly needed to tolerate the assumed fault model are employed, and the adjudicator chooses uniformly at random h results among the n provided by the variants at each execution, and then performs the simple vote only with them. The variants have no feedback about whether their result has been selected or not. Those variants whose result is not selected are called *non-participating variants*. Notice that there are $\binom{n}{h}$ possible ways to select h results among the n available.

The idea at the heart of this redundancy architecture is that choosing uniformly at random the participating variants makes it impossible for the attackers to know if the

resources invested in compromising a specific variant are wasted and then, to maintain the same probability of overall attack success, the attackers are required to invest on average more resources (other probability distributions could provide a clue about the defender strategy, thus it is better to avoid using them). Of course, this enhancement against attackers is paid in terms of additional variants that need to be fully operative (so more expensive than simple camouflage elements); this number of extra variants ($n - h$) can vary and a quantitative analysis is needed to operate a suitable choice that results in a good tradeoff among contrasting aspects (dependability assurance and implied cost, depending on the criticality of the application).

3.3.3. Deterministic Participation combined with rejuvenation

The proposed deterministic strategy (iNVP-D) is similar to iNVP-R, but the choice of the non-participating variants is made deterministically instead of randomly. In addition, it is equipped with a rejuvenation, which brings the positive effects briefly discussed in Section 3.1. The idea is that a number $n - h$ of variants is selected at each execution as excluded by the final voting, and identity of such non-participating variants changes from one execution to another. In such an organization, the higher knowledge from the attacker's perspective is countered by the defender's power to accomplish rejuvenation of variants, so that those participating to the vote include those more recently rejuvenated.

Of course, rejuvenation can be profitably applied to variants also in the previous iNVP-R strategy. However, since the choice of the participating variants is random, rejuvenation cannot be fully controlled to bring the highest benefit. Possible alternatives include to randomly choose the variant(s) to rejuvenate, or to sequentially rejuvenate variants according to some predefined order. Which one would bring higher benefit can be assessed through an analysis carried out for this purpose?

As an example, consider the architecture depicted in Figure 14 that comprises 6 variants and can be representative of both iNVP-R and iNVP-D. In the figure, the snapshot of an execution is shown, where variant 6 is under rejuvenation, and, among the five remaining in service, only the results coming from $h = 4$ of them are considered by the voter (variant 2 does not participate). The voter is assigned the protection level L_0 (highest protection level, given the higher criticality of the voter with respect to the other components), while the variants and communication channels are assigned the protection level L_1.

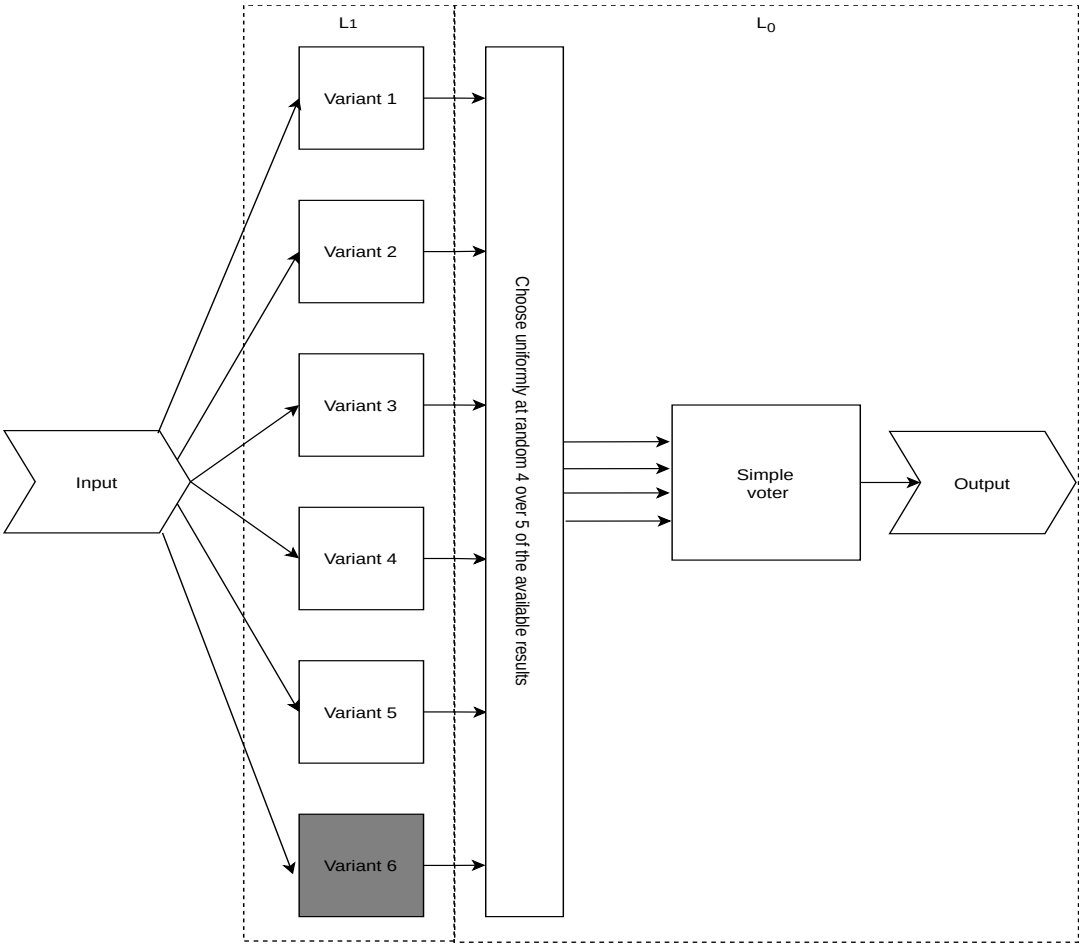


Figure 14 - Snapshot of an execution of either iNVP-R or iNVP-D, where among n=6 variants only h=4 participate to the voting (the shaded variant does not participate), and 1 variant is under rejuvenation (dark gray)

3.3.4. N Version Programming with distributed voter

As evident from Figure 13, the adjudicator is a single point of failure in the architecture. One way to protect it is to put it in a highly restrictive layer, e.g., L_0 in Figure 13. Depending on the complexity of the adjudicator, resorting to a full L_0 protection level could be infeasible/inconvenient. Therefore, an alternative, here called $iNVP_D$, is to distribute its logic in such a way that only a small kernel of the adjudication algorithm needs to be protected in L_0 and the rest can stay in L_1. For the simple voter, in [65] a distributed algorithm is presented in Figure 15, where only an interface is in L_0 and the rest of the logic is distributed among the variants, that are in L_1. Notice that it is possible to consider iNVP-R also in this case by modifying the interface logic (when a non-participating variant ask for writing its result, the interface acknowledge the writing but does not perform it).

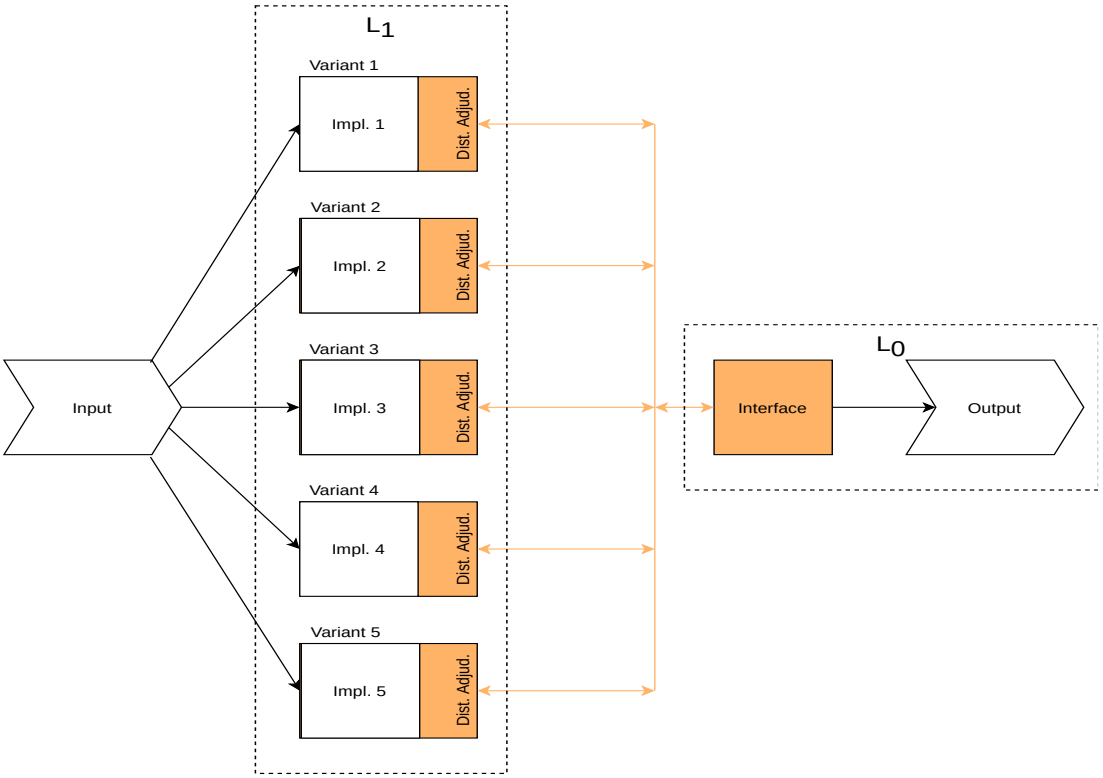


Figure 15 - NVP with distributed adjudicator, here called iNVP_D. Two access control layers (L_0, often in embedded or IoT devices implemented through a Root-of-Trust) and the adjudicator logic is distributed (orange)

The interface is designed to be sufficiently small and simple, deployed in an embedded system or for IoT devices, hosted within a Root-of-Trust, and in any case to be formally verifiable (through model checking, theorem proving, etc). Notice that this architecture is the one, among all discussed in our work, which requires less objects under L_0. Usually, fault-tolerance architectures are designed in layers [68], where components at one layer abstract their details and offer APIs to the layer immediately above it, so the adjudicator can be implemented exploiting well-known principles, tools and tweaks elaborated over the decades in the distributed system community, or even through off-the-self components. Notice, though, that exposing part of the adjudicator's logic to higher risk of attack is not recommended in all circumstances (e.g., is not recommended for a safety-critical component).

3.4. Family of RB-like Architectural Proposals

This section is dedicated to the family of redundancy architectures that follow the Recovery Blocks (RB) organization. After a brief description of the classical RB fault tolerant architecture [55], a solution obtained from its adaptation in the context of intrusion tolerance is discussed.

3.4.1. The reference Recovery Block with n variants

The basic schema of Recovery Block (RB)[7] [55] is depicted in Figure 16. In its logical organization, the n variants constituting the RB are executed sequentially, according to a predefined order. The first variant is typically called *primary alternate*, followed by the *secondary alternate* if only two variants are employed, or *second alternate*, *third alternate* and so on in case of multiple variants. The adjudicator takes the form of an *acceptance test* (AT), applied to each individual result provided by the primary or an alternate (just one AT can be employed, or each variant is associated to a specific AT). On entry to a recovery block, the state is saved to permit backward error recovery (i.e., to establish a checkpoint). The primary is executed first, and then its AT checks the produced result. If the check is successful, the RB terminates its execution by releasing this (assumed to be) correct outcome and the taken checkpoint is deleted. Otherwise, the first alternate is executed after restoring the state to the taken checkpoint, repeating the AT on the obtained result, and so on, until a successful check is encountered (RB terminates with a judged to be correct outcome) or all the alternates have completed their computation (RB terminates with a default outcome, or just a notification that no correct outcome was found). Of course, this schema is fully recursive, meaning that a variant can in turn be implemented following the recovery block structure.

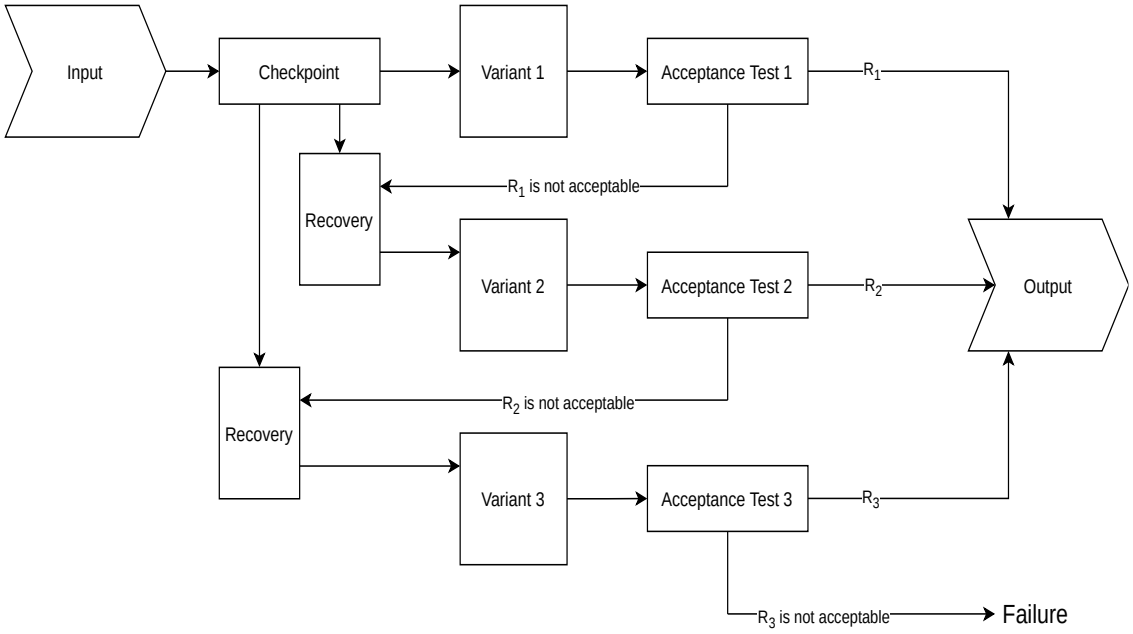


Figure 16 - Basic configuration of n Recovery Blocks (RB) employing n=3 variants

Notice that, when dealing with replicas, a single acceptance test is sufficient, but the presence of diversity among variants, as advocated for intrusion tolerance purposes, may require that a specific acceptance test is employed for each variant, depending on the resulting degree of diversity. In fact, as already discussed, correct variants may

[7] The equivalent in hardware only contexts are “stand-by sparing” or “passive redundancy”.

produce different but equally acceptable results, and RB alternates typically differ in terms of execution speed and (degradable) accuracy.

As for the NVP adjudicator, here the acceptance test is a crucial component. On the one hand, the acceptance test must be simple enough to assure higher correctness than the variant it checks, but on the other hand not so trivial as to ignore the variants' specificities and guarantee significance of the performed check. Coverage of an acceptance test, such that reliance can be put on the result of its check, depends on the application domain it is called to operate upon. Therefore, resorting to an RB structure strongly depends on the availability of acceptance tests characterized by enough coverage.

With respect to NVP, RB can be more efficient in computing resources, since in most cases only the primary is executed, while NVP exercises all the variants. However, this advantage poses also an additional implementation problem: how to synchronize the internal states of alternates that performed executions with those that did not. In fact, while the sequential execution paradigm of the RB variants (possibly involving a subset of the variants only) is fully adequate in case of stateless components, a problem arises when the variants exploit their internal state in the computations they perform over time. In this latter case, synchronization at state level is needed, to assure consistency of the computations. Parallel Recovery Block, where the primary and all the alternates are executed although only a subset of them would be strictly needed to assure termination of the RB execution, is a simple although costly solution to cope with consistency of alternates' internal state.

3.4.2. Intrusion Recovery-Block

RB for intrusion tolerance purposes (iRB) requires high protection of crucial elements, so a first simple solution consists in enclosing the checkpoint update/restore mechanism, the acceptance test and the switch that selects in turn the alternates under the protection level L_0, and the variants in L_1, as depicted in Figure 17 (similar to what proposed in [49]).

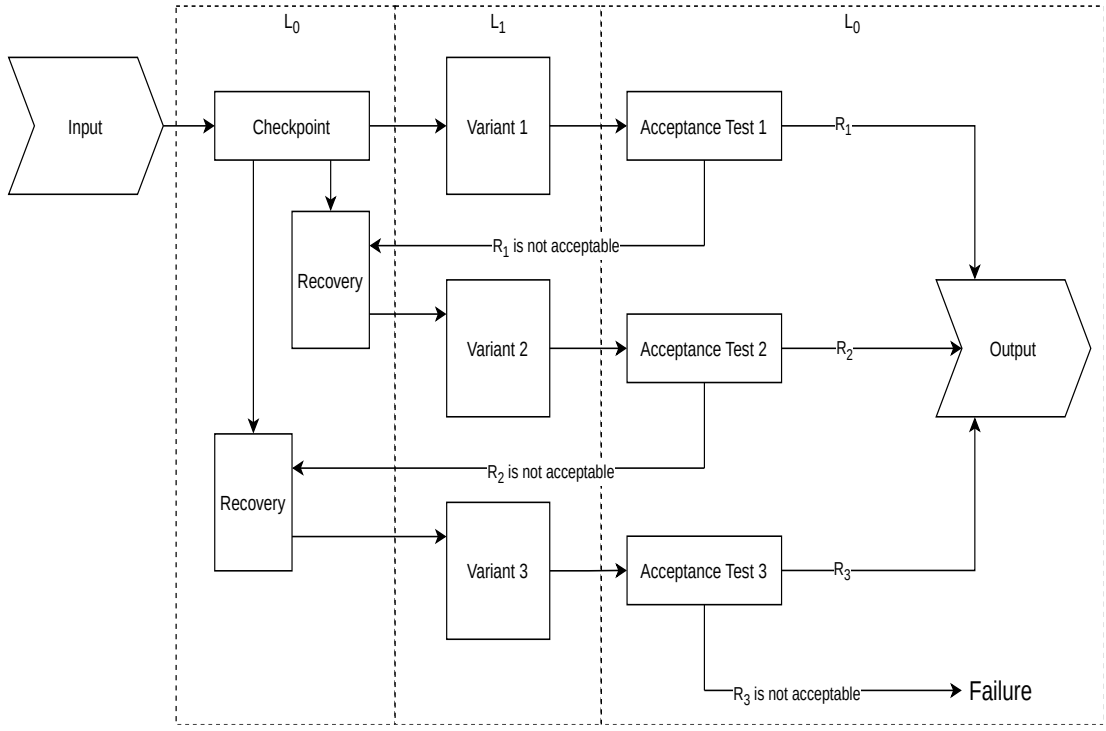


Figure 17 - Basic iRB configuration, employing 3 variants and 2 protection layers

An alternative to the redundancy architecture depicted in Figure 17 is placing also the first alternate in L_0. This is more expensive but guarantees that the most crucial variant from the attack perspective is adequately protected. Notice that the checkpoint mechanism, the acceptance tests and the switch need to be in L_0, otherwise the attackers can alter the scheme output even without compromising the variants.

To contrast the potential ability of an attacker to monitor the communications between the variants and the switch to identify the primary alternate (reading the content is unnecessary, only knowing the sender and the receiver is enough), the parallel execution of all the alternates appears as another suitable solution. In fact, it increases attacker's confusion and also saves in overall execution time in case the primary fails the acceptance test, but requires more execution resources than the pure sequential execution.

As for the intrusion tolerance alternatives based on the NVP scheme, also in the case of the iRB scheme the random selection helps in increasing attacker's confusion, while rejuvenation phases enhance the *health* of the alternates, so as to nullify the effort previously made by the attacker to compromise the rejuvenated variant.

A more protected alternative, called here iRB-R, the primary is made redundant (n_{RD} variants) and all the variants are executed, but the results of only one, selected at random, is verified by the acceptance test. This strategy is inspired by the Random Dictator approach described (Kwiat et al. 2010), and is applicable when available alternates have comparable performance and accuracy levels (to avoid penalizing quality of service indicators perceived by the user of the redundant structure).

The Random Dictator approach [69], where one variant among the h is selected at random as the primary alternate every time a result is needed, as depicted in Figure 18, exploits this direction.

Regarding rejuvenation, it can be selectively performed to have the most recently rejuvenated alternate to act as the primary at each execution. To partially compensate the cost of rejuvenation operations, a simpler RB structure composed of only the primary and secondary alternates would be sufficient, provided that: the two alternates have similar performance and accuracy levels, and the time needed to rejuvenate an alternate is shorter than the time needed to the other alternate to process an input.

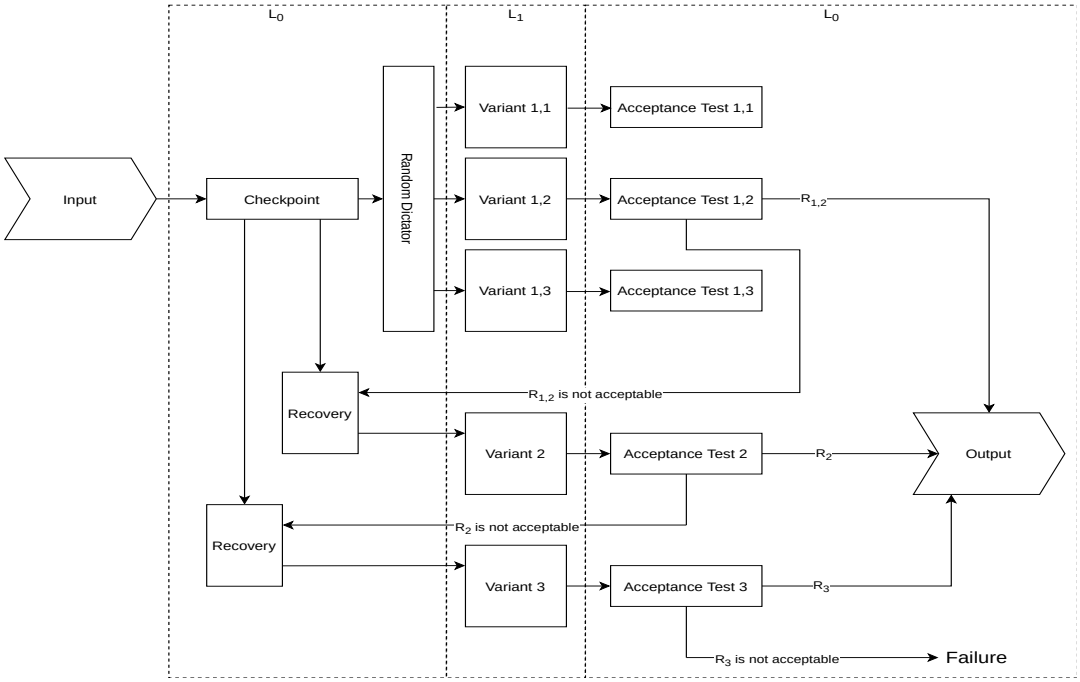


Figure 18 - Snapshot of an execution of iRB-R with 2 protection layers and 3 alternates, where the first alternate is selected uniformly at random among 3 candidates following the Random Dictator scheme

As for the previous family of NVP-like techniques, it is clarified that the just discussed RB-like solutions are not meant to be an exhaustive intrusion tolerance set; rather, they are examples of how the features discussed in 3.1 can be exploited to adapt the RB tolerance organization to cope with intentional attacks. So, there is room to investigate other interesting alternatives.

3.5. Family of Hybrid Architectural Proposals

NVP and RB are recognized as the two extremes of redundancy-based fault tolerance techniques: exploitation of maximum execution resources to achieve minimum execution time (NVP) and minimum execution resources to be potentially payed by maximum execution time (RB). In between, hybrid solutions that try to combine the best aspects of each of the two have been proposed in the literature. Three of them have been selected, briefly recalled in the following (SCP [70], CRB and SCOP [71]), and for each of them an alternative suitable to address intrusion tolerance is presented.

3.5.1. N Self-Checking Programming

The N Self-Checking Programming (SCP)[8] architecture consists in the parallel execution of n_{SC} self-checking components, ordered according to some criteria (typically, based on performance and accuracy considerations). The outcome of the NSCP structure is the result provided by the first self-checking component, starting from the first one in the ordered list. A self-checking program results from the addition of redundancy into a program to check its own dynamic behaviour during execution [72]. As reported in [70], a self-checking component consists of either a variant and an acceptance test or two variants and a comparison algorithm.

In Figure 19, a NSCP configuration is depicted, where four variants are involved and organized in two self-checking components, each one resulting from the association of two variants with a comparison algorithm, such that an output is produced only if the comparison between the results of the two variants is successful.

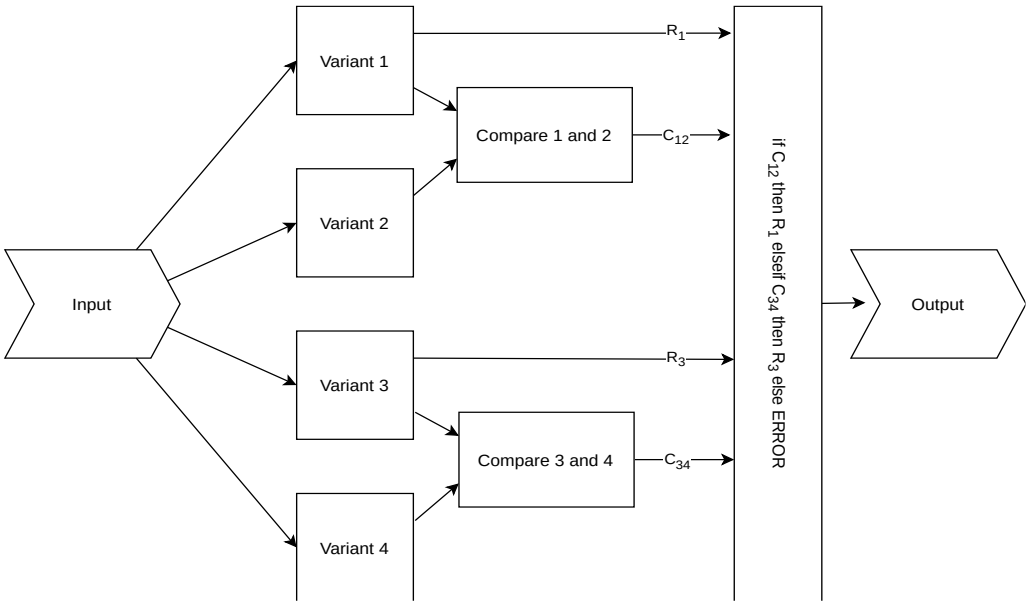


Figure 19 - Example of SCP configuration with two self-checking components, each one exploiting two variants and a comparator

3.5.2. Intrusion Self-Checking Programming

Borrowing ideas from intrusion tolerant NVP and RB, a basic architecture for the intrusion tolerance counterpart of SCP, called iSCP, consists in exploiting different protection layers to enhance defence against attacks.

[8] The equivalent in hardware only context is “active dynamic redundancy”.

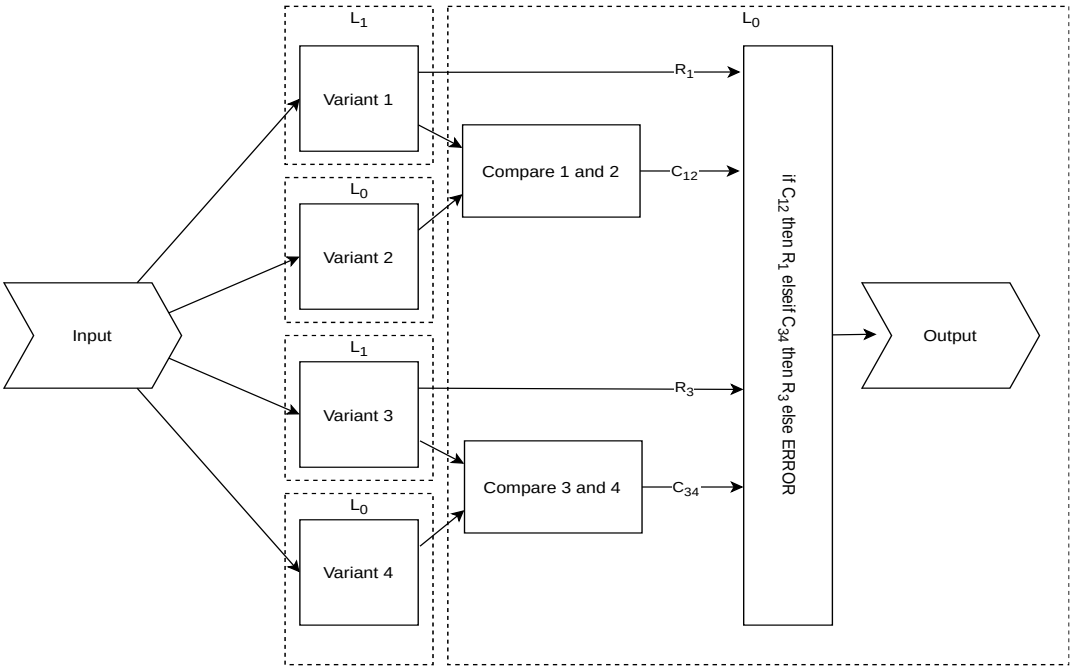


Figure 20 - iSCP with 2 protection layers and 4 variants grouped in 2 self-checking components

Notice that the configuration in Figure 20 guarantees tolerance of 1 arbitrary intrusion, and 2 intrusions only if manifested with non-coincident errors. When tolerance to multiple arbitrarily compromised variants is needed, the approach of iSCP is to define a self-checking component as in Figure 19, where at least one variant is assigned the highest protection level L_0, as for the comparator component. This is because the common mode failure between the two coupled variants is expected to be not a rare event when intentional attacks are considered, and so a phenomenon that need to be mitigated from the security perspective (as also pointed out in [49]). In this case, it is relevant to distinguish a value failures due to accidental causes from i value failures due to intentional attacks, so that $f = a + i$, and assume that the wrong results produced by accidental causes are different from the ones produced by intrusions (otherwise the attacker needs to read the accidentally wrong value). Since variants in L_0 are protected against attacks, but can still suffer from an accidental fault, tolerance abilities of this architecture are: $a \leq 1$ and i that depends on n and a as reported in Table 4 (within a self-checking couple, at most one variant can be affected an accidental fault and at most one by an intrusion).

In case the self-checking component results from a couple *variant and acceptance test*, for the same reason discussed above it is appropriate that the acceptance test receives a higher protection (L_0).

An alternative iSCP architectural solution able to tolerate the presence of f compromised versions would be to increase the redundancy within a self-checking component (i.e., each self-checking component comprises $g \geq 2$ variants, whose results are submitted to a majority voter), resembling an iNVP structure, to enhance its tolerance ability. Unfortunately, the resulting number of needed variants is significantly higher than for the corresponding iNVP alternative with the same tolerance abilities, so not competitive and therefore not considered in this study (see Appendix II for details). Confusion strategies could be further added but are not investigated here.

3.5.3. Consensus Recovery Block

The Consensus Recovery Block (CRB), depicted in Figure 21, reduces the importance of the acceptance test used in RB and is able to handle the case where NVP does not employ a sophisticated voter able to recognize multiple correct outputs [55]. In CRB the variants are ranked, and, on invocation, all variants are executed in parallel, and their results submitted to a voter. The original formulation of the scheme [73] assumes that there are no common mode failures, so erroneous results do not coincide. Therefore, agreement between the outcomes of two variants is sufficient to deliver this value as the final result. However, in a more general formulation, which is comprehensive of less restrictive failure model assumptions, the voter can be based on a simple majority (so the architecture can tolerate $f = m - 1 = \lceil (n-1)/2 \rceil$) or another plurality criterion to consider an outcome to be successful. If there is no majority, then the result of the variant with the highest ranking is submitted to the corresponding acceptance test. If this fails then the next variant in the order is selected. This continues until all variants are exhausted or one passes the acceptance test.

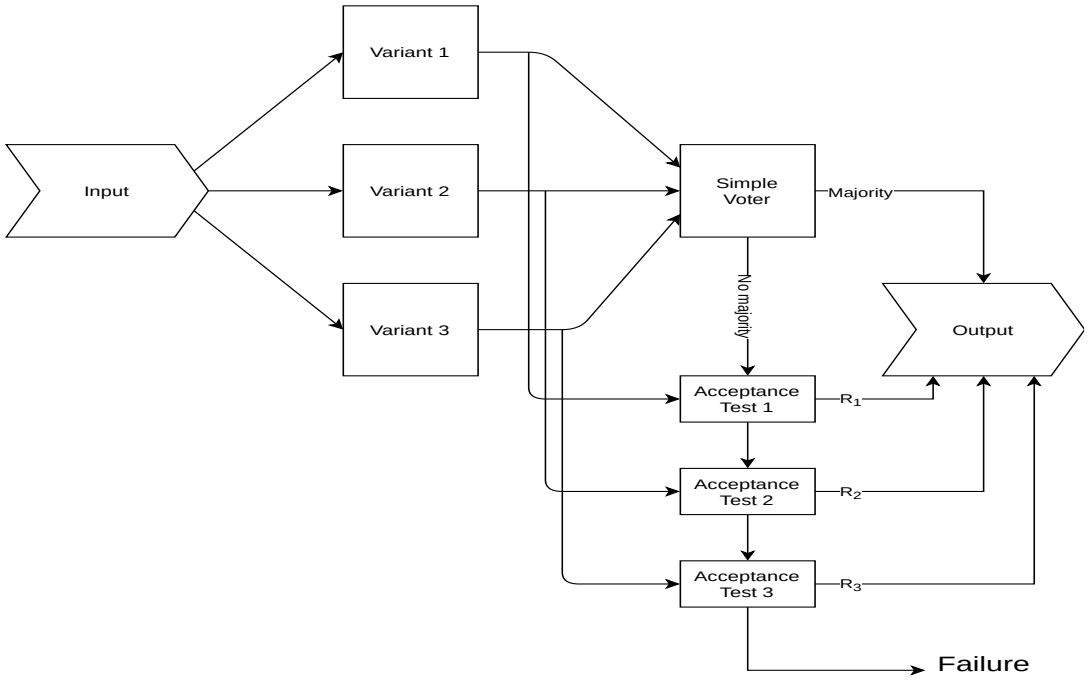


Figure 21 - Basic configuration of the Consensus Recovery Blocks (CRB), instantiated for n=3 variants

Notice that this schema is, on one hand, a parallel recovery block with a pre-test about consensus, and on the other hand an NVP with an adjudicator that is more sophisticated than the simple voter.

3.5.4. Intrusion Consensus Recovery Block

As for the other proposed intrusion tolerant alternatives to basic fault tolerance strategies, also for an intrusion version of CRB (iCRB) a first measure to adopt is higher protection of the most critical components of the scheme, i.e. the implementation of the two-step logic (voter and acceptance test, which are assigned protection level L_0) with respect to variants (which are assigned protection level L_1).

Since CRB is a hybrid between NVP and RB, protection techniques already discussed when presenting intrusion tolerance alternatives of NVP and RB could be considered for iCRB proposals. In particular, the architecture depicted in Figure 22 is suggested, where confusion aspects obtained through addition of extra variants whose outputs are not

considered by the voter component are exploited. If the employed variants have degradable quality of service, the added non-participating redundancy could be inserted in the ranking in different position from one execution to another. Instead, if comparable variants are employed, the outputs considered by the voter can be randomly chosen at each execution, to enhance the attacker's confusion level. Then, in case the voting phase is not successful and acceptance tests are activated, the output of previously non-participating variants can be checked by the respective acceptance test (provided they are available) or not, depending on the degree of reliance that can be put on them.

If affordable from the overall budget perspective, the presence of extra redundancy favours the usage of rejuvenation actions, as a further protection measure, with expected benefits as already previously discussed. In addition, changing the logic of the overall adjudication function to have the acceptance test applied to the result selected by the voter, in case this happens, strengthen the scheme to a greater extent. In fact, taking advantage of the availability of both the voter and the acceptance test, making such double checks enhances the chance to counteract potential intrusions.

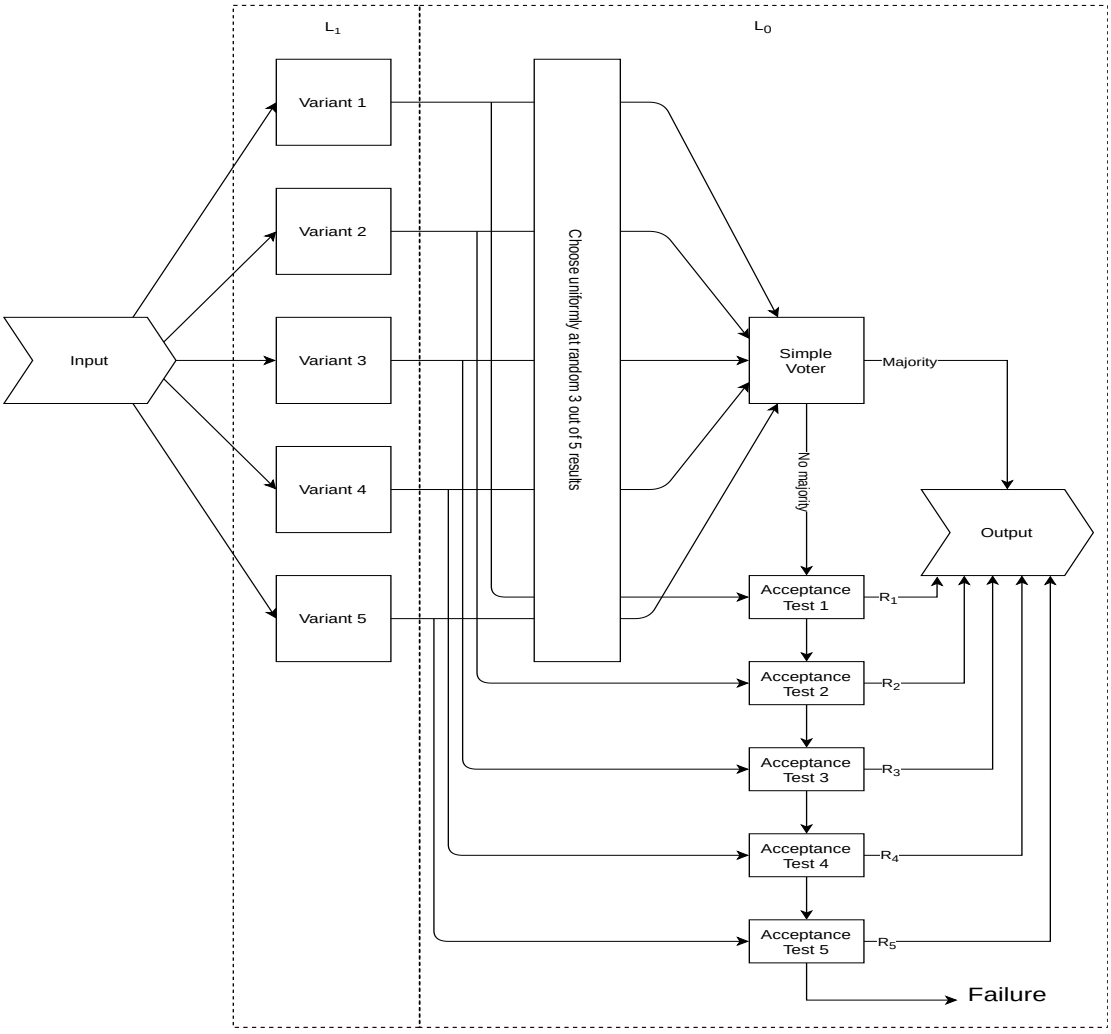


Figure 22 - An iCRB configuration, obtained adding 2 protection layers and extra variants for confusion (among the n=5 variants, only h=3 participate to the vote, but all the 5 are checked by the acceptance tests)

3.5.5. Self-Configuring Optimistic Programming

With the aim of improving the cost-effectiveness of fault-tolerant software (diminishing the waste of resources) in [71] the Self-Configuring Optimistic Programming (SCOP) has been presented. The idea is to maintain the logic of NVP unaltered but schedule the

execution of the variants in phases, instead of the parallel execution of all the variants, to promote efficiency. The scheme is based on an optimistic vision, since high quality versions are typically employed to build redundant organization for critical domains. The idea is to start executing the minimum number of variants that, if all correct, satisfy the adjudicator criterion and the scheme terminates. If this is not the case, a new execution is started, involving the minimum number of variants among the ones remaining to be executed, such that, if successful, will contribute together with the variants already executed in the previous phases to satisfy the adjudicator criterion and terminate the overall execution. This pattern is repeated until a successful result is found, or all the variants are exhausted.

The example in Figure 23 helps to figure out how SCOP works; more details are in [71].

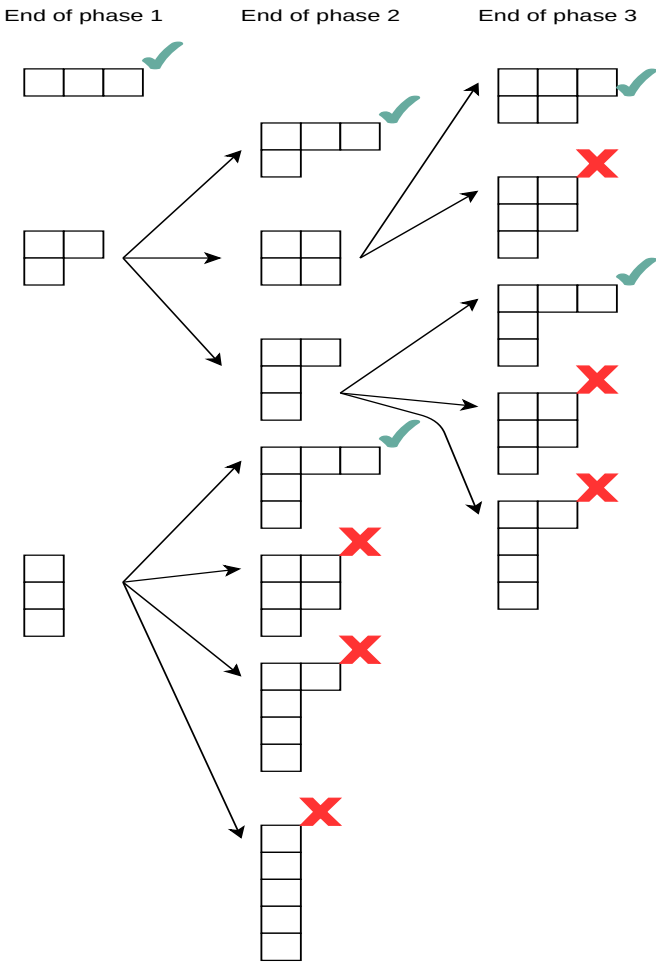


Figure 23 - Example of SCOP for n=5. Here Young diagrams are exploited to represent agreement among variants (squares represent results, same row represent agreement, the rows are ordered according to the number of agreements)

Consider $n = 5$. The majority is $m = 3$, so in the first phase only 3 variants are executed, and their results passed to the adjudicator. If there is a majority, i.e., the three results agree, then the result is sent in output without executing the other two variants. If there is no agreement among the three variants of the first phase, but there is agreement among two of them, then in the second phase another variant is executed, and its result compared with the already available ones. A third phase maybe needed involving the last available variant, if the result of the variant executed at the second phase does not contribute to obtain a majority value. Instead, if the three variants of the first phase produce three different results, then two variants are executed in the second phase.

Therefore, this SCOP configuration most likely terminates after only one phase (involving three variants), but in the worst case may need three phases, involving all the variants.

3.5.6. Intrusion SCOP

The introduction of access control layers follows the same approach as in Figure 15. The novelty relies on the fact that the very nature of SCOP promotes the application of confusion. In fact, at the beginning of each phase, at least two strategies are feasible:

- execute more variants than needed and select uniformly at random a subset of results of required cardinality.
- select at random just the required number of variants, choosing among a surplus of available ones, and execute only them.

In both cases, the attacker is not able to precisely determine which are the m variants whose results are considered by the voter. Notice that the former, when forcing SCOP to comprise a single phase, is equivalent to iNVP-R. The latter produces even more combinations with respect to SCOP, as shown in Figure 24, for a specific case.

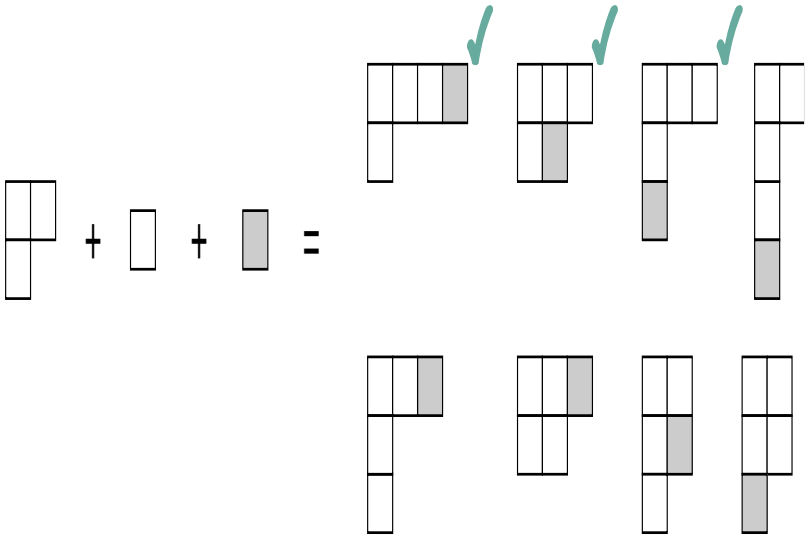


Figure 24 - Example of the possibilities in the second phase of iSCOP (of the first kind) if m=3, knowing the configuration at the end of the first phase and assuming that, instead of one, two variants are executed and one of the results does not participate (in gray in the picture). Notice that in 2 cases out of 8 listed accepting both new results would have allowed to stop in the second phase

As for iNVP, choosing variants at random is not always appropriate when introducing also the rejuvenation process. An alternative is designing a more complex adjudicator that selects the value to send in output working on more informative syndromes. For instance, being not all the variants' results required in the first phase, a subset of variants can be rejuvenated, and the last rejuvenation time of a variant can help in deciding whether its result is reliable or not, so it is useful to include it in the syndrome.

3.6. A practical summary of redundancy-based Intrusion Tolerance schemes

To better support the understanding and selection of the appropriate intrusion tolerance scheme to adopt for specific purposes, in the following the solutions introduced in the previous sections are schematized from a practical perspective. In particular, indications about redundancy levels, degrees of confusion and of variants under rejuvenation, as well as relation with available sites are provided.

First, Table 2 reports the number of variants that are needed by each of the five fault tolerance schemes considered in this study, to tolerate f (in the worst case common-mode) value failures and k omission failures, occurring simultaneously. Of course, f or k can be 0, in case only omission failures or only value failures are assumed, respectively. Also, indication about the kind of decision function adopted by the scheme is included. For NVP, the simple majority voting is assumed, and for SCP the self-checking component is obtained through comparison of two variants' outcomes. Observe that, when only omission failures are considered, the decision function based on voting is simplified to be just the selection of the received variant's value (in accordance with the omission failure assumption, if a variant output is issued, it is correct). Moreover, similar formulations can be easily derived for determining the number of required variants for NVP and SCP if a different voting function or a different realization of a self-checking component than considered in Table 2 are adopted, respectively.

Table 4 is dedicated to the new proposed intrusion tolerant alternatives to the schemes in Table 2. They are recalled in Table 3. These schemes take advantage of additional features to better cope with intentional attacks, as deeply discussed in Section 3.2. Specifically, they consist in: i) additional redundancy used as a stratagem to confuse the attacker; ii) distribution of the variants on more sites; and iii) periodic rejuvenation of variants, to contrast potential partial compromise of a variant already in place, or anyway to nullify potential gathered knowledge by an attacker about a variant. As previously introduced, h indicates the number of variants considered by the adjudicator (therefore, $n-h$ indicates the number of additional redundancies for confusion), s indicates the number of available sites, and r indicates the number of variants under rejuvenation. The formulas in Table 4 for the number of variants required by each scheme include these parameters h, s, r, in addition to f, a, i, k connected with the failure types.

The central information shown in Table 4 are the number n of needed variants to tolerate $f+k$ failures, expressed in terms of f, k, s and r, and considerations about additional redundancy for confusion. Regarding the latter, there is no exact indication on the amount of extra redundancy needed, since this choice is left to the system designer. What is expected is that a higher number of extra redundancies for confusion should correspond to a higher defence ability (and therefore higher dependability); however, this needs to be confirmed by quantitative analysis, that is planned as a future research study. So, it is only indicated that, if the total number n of variants grows with extra variants, the needed number variants for tolerating $f+k$ failures reported on the left column represents the number of variants whose results are considered by the adjudication function, that is h. Additionally, the population of variants under rejuvenation is chosen by the system designer, trading between the cost of rejuvenation and benefits in prolonging the life of correctly operating variants; so only its number r is accounted for in the formulas.

iNVP with simple majority voting requires

$$n = 2f + k + 1$$

variants to tolerate f (in the worst case common-mode) value failures and k omission failures, being the majority $m = \lceil (n+1)/2 \rceil$, as reported in Table 2.

If the defender has s sites, the best strategy is to distribute as uniformly as possible the variants among the sites. Thus, there are $\lceil n/s \rceil$ variants on the largest site, and then $k \geq \lceil n/s \rceil$. Applying standard properties of ceil function (details are in Appendix I) it is possible to relate n directly to f and k (datum), and s and r (designer choice), as in Table 4. However, the uniform distribution is not a compelling requirement, so other deployment policies can be adopted. As a general rule, the necessary condition to prevent the occurrence of a system failure, following the isolation of one site by an attacker, is that less than the number of variants whose results are needed to satisfy the adjudication function are allocated to any single site (a majority of variants, in case a majority voting is employed in the scheme, as for the case presented).

When confusion is adopted, $h = 2f + k + 1$ and $n \geq h$ (i.e., the total number of variants is always greater than the participating ones), in both deterministic and random strategies.

iRB requires a smaller number of variants, $n \geq f + k + 1$, with respect to iNVP and n does not change when the isolation of a site is considered as long as $n > s$, that is usually the case. For degradable systems, where the variants in iRB are ordered according to results' accuracy, the best strategy to distribute the variants among sites is to deploy the primary on one site, the second alternate on another site, and so on till the first s variants are assigned and distribute the remaining $n - s$ round robin among the sites. Confusion is mainly applied to the primary, and, calling n_{RD} the number of variants in the iRB, implies that $n \geq f + k + n_{\mathrm{RD}}$. In this case, the $n_{\mathrm{RD}} - 1$ additional variants have to be deployed on different sites.

Table 2 - Comparison of classical architectures. k is the number of omitted results. Legend: arch=architecture, n=number of variants, decision=decision mechanism

arch	n	decision
NVP	$2f + k + 1$	Relative, simple majority
RB	$f + k + 1$	Absolute, based on ATs
SCP	$2(f + k + 1)$, with $f \leq 1$	Relative, compare 2 results
CRB	$2f + k + 1$	First relative and then absolute
SCOP	$2f + k + 1$	Relative, simple majority

For iSCP, when $f = 0$, to tolerate k omission failures, that in the worst case are distributed one per couple in $n_{\mathrm{SC}} - 1$ self-checking components, $2(k + 1)$ variants are required. If in addition there is $f = 1$ value failure, then the required number of variants became $n = 2(k + 1 + f)$. To tolerate $a \leq 1$ accidental and i intentional failures, iSCP then requires

$$n = 2(k + 1 + a + i).$$

Table 3 - Acronyms

Acronym	Full name	Section	Figure
iNVP	intrusion N Version Programming	3.3	Figure 12
iNVP-R	iNVP with Random Participation	3.3	Figure 14
iNVP-D	iNVP with Deterministic Participation	3.3	Figure 14
iRB	intrusion Recovery Block	3.4	Figure 17
iSCP	intrusion Self-Checking Programming	3.4	Figure 20
iCRB	intrusion Consensus Recovery Block	3.4	Figure 22
iSCOP	intrusion Self-Configuring Optimistic Programming	3.5	Figure 24

iCRB comprises two phases, but to determine the number of variants only the first phase, where iCRB behaves as iNVP, is relevant. Thus, $n \geq 2f + k + 1$. When considering s sites, the number of required variants is reported in Table 4. For confusion in iCRB, even though

the results of those variants that do not participate to the vote are considered in the second phase, being $n \geq h \geq 2f + k + 1 \geq f + k + 1$, the same reasoning as for iNVP applies.

Table 4 - Intrusion add-ons

Scheme	n	Confusion
iNVP, iNVP-R and iNVP-D	$2f + \max\{k, \lceil\frac{2f+1}{s-1}\rceil\} + r + 1$ where $s > 1$	Naturally integrable. Substitute h for n
iRB	$f + k + n_{\mathrm{RD}}$ if $n > s$	Mainly applied to the primary
iSCP	$n = (a + i + \max\{k, \lceil\frac{2a+2i+2}{s-2}\rceil\} + 1) + r$, where $s > 2$ and $a \leq 1$	Not investigated
iCRB	$2f + \max\{k, \lceil\frac{2f+1}{s-1}\rceil\} + r + 1$ where $s > 1$	Same as NVP
iSCOP	$2f + \max\{k, \lceil\frac{2f+1}{s-1}\rceil\} + r + 1$ where $s > 1$	Naturally integrable. Several options

Finally, considerations about performance are summarized in Table 5. Since the execution logic of the intrusion tolerant alternatives is the same as the original fault tolerant schemes from which they derive, the table is based on the original schemes. In fact, the intrusion tolerance ability is mainly obtained through additional redundancy and protection measures, and the execution time may result longer due to the impact of these additions, but the execution model is unchanged (that is, sequential of parallel execution of the variants).

Without going in the detail of a huge variety of system organizations and application domains, the time requirements are abstracted at the level of *hard time constraints* and *soft time constraints*. The former indicates that violation of the time requirement has potentially heavy consequences for the system where the scheme is embedded, while the latter indicates a lower criticality of the time requirement. Therefore, roughly it can be suggested that schemes based on parallel execution are adequate for hard time constraints, while schemes structured in sequential phases are risky from the hard time perspective. However, this is an indication, but not a definitive discrimination among the considered schemes. Indeed, while parallel execution allows to predetermine the worst-case execution time of the slowest variant and so be sure of the maximum time required by an execution of the scheme, mechanisms structured in phases have variable execution time depending on the failures really experienced during the execution (they afford longer execution time in unfavourable scenarios, but save in executed variants in the more frequent favourable scenarios where no failures occur). However, also for these sequentially based solutions, the worst-case execution time can be computed and, if adequate for the hard time constraint imposed by the application at hand, there is no objection on adopting one of them.

Of course, when soft time constraints are in place, any of the presented schemes can be applicable, and the choice will be in general operated in accordance to some other criterion.

Table 5 - Comparison of the architectures with respect to time constraints

Arch	Hard time constraint	Soft time constraint
NVP	OK (parallel exec.)	OK
RB	KO (sequential exec.)	OK
SC	OK (parallel exec.)	OK
CRB	OK (parallel exec.)	OK
SCOP	KO (sequential exec.)	OK

3.7. Redundancy-based intrusion tolerance from the different system components' perspective

This section concentrates on discussing the redundancy-based intrusion tolerance solutions, developed in 3.3, 3.4 and 3.5, from the perspective of the different system components of an ICT system, to which such schemes are intended to be applied.

Recalling from [74], the ICT components that can be considered the target of a cyberattack are grouped in the following three categories:

- *Computing element*, which is a component that is devoted to performing some kind of functionality, to provide a service to the requesting entity (a user or another component). Operating systems primitives, software applications and enterprise software are typical examples of this category.
- *Communication element*, which is the means through which information is delivered to/from computing elements, users and storage. The internet and the several wireless networks technologies are typical examples of this category.
- *Data storage element*, which includes different storage technologies used to retain digital data within a computer system architecture. The term storage may refer both to a user's data generally and, more specifically, to the integrated hardware and software systems used to capture, manage and prioritize the data. This includes information in applications, databases, data warehouses, archiving, backup appliances and cloud storage.

These three component categories are characterized by hardware/physical supports and software programs, either devoted to performing specific functionalities (computing element category) or to manage/control the operation of the hardware/physical support (communication and data storage categories).

It is underlined that the interest in this work is on cyberattacks, so the impact of an attack on a physical component can only occur through the software facilities that control/act on it. Direct physical attack to corrupt a portion of a physical medium (as it could be a memory cell or sector) is considered out of scope.

Following this observation, in principle any of the proposed redundancy-based intrusion tolerance schemes would be adequate for enhancing resilience of ICT components belonging to the three categories, considering the aspects discussed in Section 3.1 to support the most suitable selection among the several alternatives. However, while functional components employed at application level are typically developed as ad-hoc components to accomplish the activity the application is called to perform, the software supporting the operation of physical devices, as well as operating systems, libraries and the execution environment are typically off-the-shelf components. This implies that, to obtain the diversity advocated to be a fundamental aspect characterizing redundancy-based intrusion tolerance, full control by system developer is possible for in-house developments, while for the other software components the only option is to rely on what is available on the market. Luckily, there is a wide range of options made available by ICT companies, each one embedding some peculiar aspects that make their products equivalent from the service point of view, but with differences in terms of how such service is provided. Open-source repositories also help significantly, especially for what concerns libraries and execution environments. Therefore, the diversity principle the intrusion tolerance schemes are based on can be easily satisfied. Moreover, resorting to employ a variety of pairs of *physical devices, managing software*, as it would be for communication networks and data storage components, enhances system resilience also against faults affecting the hardware part.

Concluding from this discussion, it can be inferred that the presented redundancy-based intrusion tolerance schemes can be profitably exploited to protect ICT components. The highlights elaborated in Section 3.6 help a system designer in selecting a suitable solution for the faced requirements and constraints.

3.8. Integration in ResilBlockly

The developed intrusion tolerant architectures can be profitably integrated within the risk assessment process carried on in BIECO, as one of the means to improve resilience of components evaluated as being 'unacceptably critical' from the risk analysis evaluation. In the following, the integration with the Risk Assessment performed through the ResilBlockly methodology and tool is discussed, taking a specific component of the ICT Gateway Use Case as application example.

First, the process resorting to the selection and application of the ITC to a selected component, in place of the original creation (referred as simplex component) is described.

Redundancy-based intrusion tolerant architectures are considered either when designing new components or for strengthening already existing ones. In both cases, a model of the component is built, following ResilBlockly supported formalisms. The derived model is then analysed in ResilBlockly. In this section the focus will be only on Risk Assessment, meaning that weaknesses and vulnerabilities of the simplex component are identified by the modeler (either searching in the CWE and CVE databases or manually defining them) and then their severity and likelihood are declared, as detailed in Section 3 of BIECO D6.2. In particular, redundancy-based architectures can be employed to address weaknesses (vulnerabilities are better addressed by other intrusion tolerance techniques) [67] [49]. If the risk assessment outcome is such that the simplex component requires to be made more robust in order to have the overall application fulfil the security requirement, the next step is performed.

The second step is to select one or more intrusion tolerant architectures among those detailed earlier in Section 3, or customized starting from fault tolerant architectures and following similar ideas. Which one to choose highly depends on system requirements and context of the application where the simplex component operates, but also on its identified weaknesses, in terms of severity and likelihood. In general, more than one architecture could in principle fit the application's needs. For each architecture, the number n of variants is then set, following the indications sketched in Section 3.6. Knowledge of the probability that a given weakness is exploited in the simplex component (a numerical value, more precise than likelihood) evaluated studying logs, also of similar components belonging to other applications, and adjudicator/acceptance tests coverage are assumed, as well as reliability or probability of undetected failure of the overall architecture. There is extensive literature on how to analyse a redundancy-based architecture but, with the focus of this section being on ResilBlockly, here only probability of undetected failure will be considered because it directly relates to the likelihood in the Risk Assessment. Once the selection of the ITC architecture is made, the model of the simplex component is updated to consider its new redundant version, and a new risk analysis is performed in ResilBlockly.

The third step is then to compare performance (time, computational resources, etc), severity and likelihood obtained from the analysis of the simplex component with those of the redundant counterpart analysed in the second step. This kind of A/B comparison can end up in a decision on the configuration to choose or highlight the need of redefining the number of redundant variants n and re-iterate the second step. If carefully set up, it is expected that such iteration process closes in a couple of iterations.

As a case study, let's consider the ICT Gateway detailed in BIECO D6.2 (and depicted in Figure 25) and focus on the Security & Resilience component (S&R). This component runs on a separate server and is responsible to detect malicious activities. Upon detection of a malicious activity, it raises alerts to GUI and applications, that in turn can

restart other ICT Gateway components. Thus, if S&R is intruded then it can raise false alerts that lead to a higher probability of outages or losses.

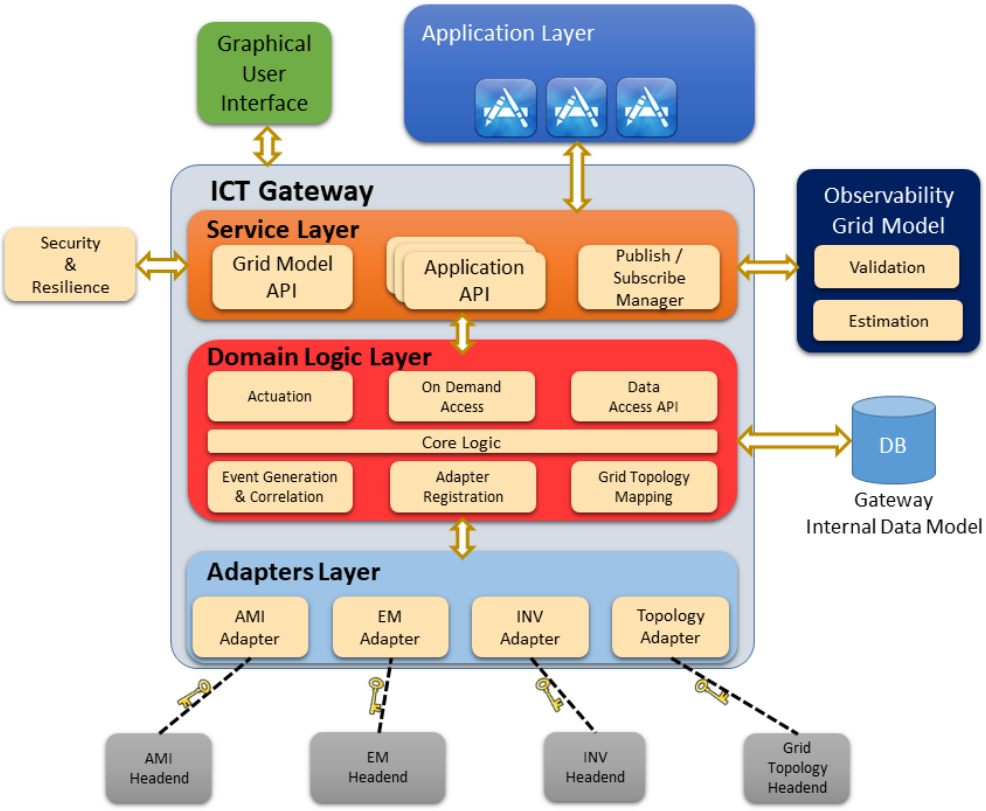


Figure 25 - ICT Gateway architecture

Starting from the model of the ICT Gateway, in ResilBlockly it is possible to list the weaknesses of the simplex S&R and indicate severity (in this case 10, the worst case) and likelihood (in this case Moderate), as depicted in Figure 26. As an example, the CWE weakness 693 called “Protection Mechanism Failure” is analysed.

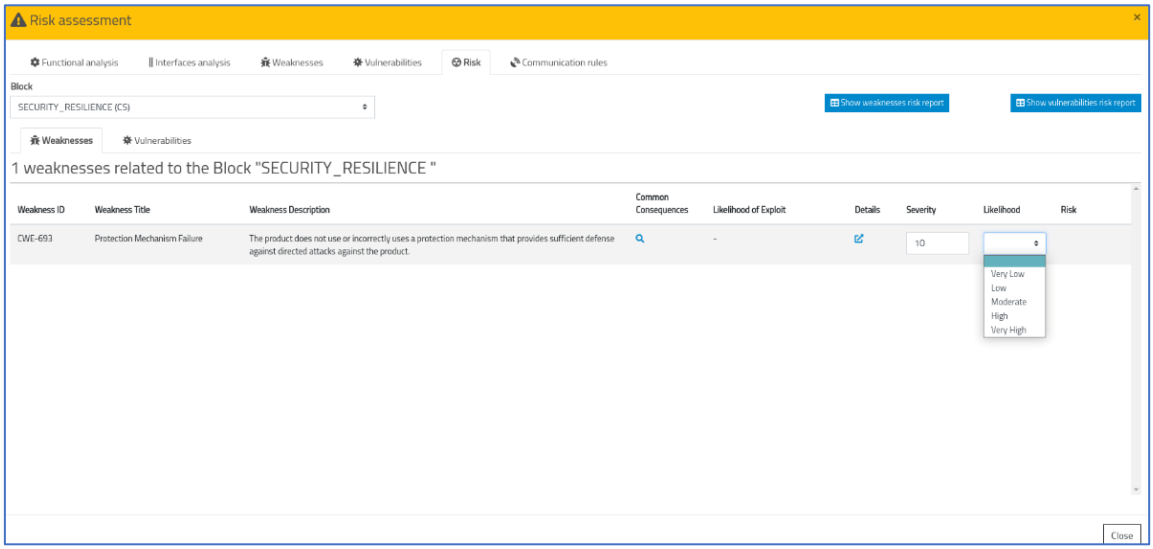


Figure 26 - Risk assessment of the simplex Security & Resilience component

Without going into the details of their selection process, iRB (Section 3.4) and iNMR (Section 3.3) are the two redundancy-based intrusion tolerant architectures to consider

for replacing the simplex version of the S&R component. Their models are then built and values to their parameters are assigned.

In order to declare the weakness likelihood, a qualitative value, it is recommended to perform first a quantitative analysis (e.g., evaluate the probability of delivering wrong results) to support the choice. Notice that the correspondence between probabilities range and likelihood highly depends on the context in which the system is deployed and has to fixed in advance. For the purpose of example only, it is assumed that $\mathbb{P}_{\text{fail}}$, is the probability of a variant failure, $\mathbb{P}_{\text{noncoverage}}$, the probability that the acceptance tests wrongly accept the results, and $\kappa \in [0,1)$, the degree of diversity of the variants are known (as estimated in other studies). In particular, the higher κ is, and the more diverse the variants are. Thus, following the simple analysis detailed in [49], $\mathbb{P}_{\text{ufail}}$, the probability of undetected failure, of iRB is

$$\mathbb{P}_{\text{ufail}} = \mathbb{P}_{\text{fail}} \cdot \mathbb{P}_{\text{noncoverage}}$$

And $\mathbb{P}_{\text{ufail}}$ of iNVP, assuming the probability of failure of adjudicator extremely low with respect to the other number involved, is

$$\mathbb{P}_{\text{ufail}} = \begin{cases} \mathbb{P}_{\text{fail}}\left(1 - \dfrac{\lfloor n/2 \rfloor}{\lfloor (1-\kappa)(n-1) \rfloor}\right) & \text{if} \lfloor n/2 \rfloor \leq \lfloor (1-\kappa)(n-1) \rfloor, \\ 0 & \text{otherwise.} \end{cases}$$

The value of $\mathbb{P}_{\text{ufail}}$ together with the information of Table 4 are exploited to compare several configurations of iNVP and iRB. Consider for instance s=2 sites, k=1 and f=1 (tolerance of one omission and one value failure), r=1 (rejuvenation of one variant at a time), $n_{RD} = 3$, $\mathbb{P}_{\text{fail}} = 0.9$, $\mathbb{P}_{\text{noncoverage}} = 0.0001$, and $\kappa = 0.45$. Then n=5 for iRB and $\mathbb{P}_{\text{ufail}}$ is about 10^{-4}, and n=7 for iNVP and $\mathbb{P}_{\text{ufail}}$ is almost zero. Thus, it is possible to update the assessment in ResilBlockly as in Figure 27, where the custom weakness for iRB is “Acceptance Tests Coverage is not Perfect, then Wrong results can be Accepted as Correct”, and “A majority of variants fail producing the same incorrect result” for iNVP, and the likelihoods are now set to low and very low, respectively. The final choice on which configuration to select is done considering performance. Indeed, assuming the information of Table 5 is enough in the context of Smart Grids to make the decision, the presented configuration of iNVP has to be selected.

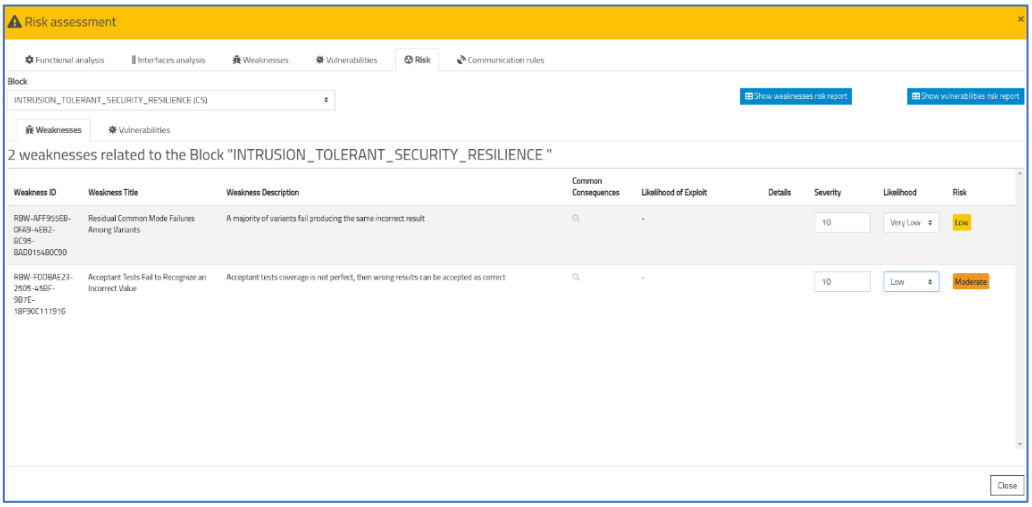


Figure 27 - Risk Assessment of iRB and iNVP

4. Mitigating Risk during Development via Assurance Case Modelling

In this section, we discuss how assurance cases that argue in terms of dependability risk mitigation can be constructed using elements from BIECO concepts and tools. In sections 2.1, 2.2 and 2.3, we discussed how guidance from standards for safety and security can be leveraged to provide a combined approach for safety and security assurance. For mission-critical applications, where different dependability properties are more relevant, the approach can be adjusted to address risk with respect to those properties instead.

4.1. Workflow Overview

The assurance case is a central artifact of this process, as it can capture the rationale arguing how the residual risk to the system's mission has been rendered acceptably low by the end of development. Our proposed approach can be described in terms of the V-model lifecycle, as described in sections 2.1 and 2.2:

1. **System definition.** In this stage, a (possibly preliminary) specification of the system and its operational environment needs to be established.
2. **Initiate Dependability Assurance Case.** The assurance case can be maintained from the early stages of development and be progressively updated as more information becomes available. The intent is to use it as a live document, which can monitor the progress of the development assurance, and coordinate the activities across the system stakeholders.
3. **Dependability Hazard Analysis and Risk Assessment (HARA).** In this stage, based on the system and environment specification, identification of the relevant events that could cause unacceptable violation of the application's dependability properties takes place. These can be referred to as 'dependability hazards. Each dependability hazard that is considered relevant for the application is then assessed in terms of the overall risk it presents to the application, in terms of its impact and likelihood of occurrence. Depending on the application domain, domain-specific risk rating systems may be used e.g., for the automotive domain safety hazards are rated in terms of their estimated severity, exposure, and controllability. Hazards may also be further refined in terms of the operational situations they could occur. The combinations of a given hazard with relevant operational situations are also referred to as **Hazardous Events (HEs)**. From this point onwards, we will refer to both hazards and HEs as HEs. The set of rated HEs can be prioritized in terms of risk, and specific HEs can be excluded if their risk is argued to be acceptably low.
4. **Dependability Goal Specification.** In this stage, goals for protecting against the violation of the dependability HEs identified as relevant and of sufficient risk need to be specified. Dependability goals are typically high-level requirements that, when implemented correctly, tolerate, mitigate, or eliminate the associated HEs. A goal may be simultaneously addressing multiple HEs.
5. **Dependability Concept Specification.** In this stage, the means necessary for achieving the dependability goals specified previously are specified. The means should be specified in terms of technologies or procedures that have been established to satisfy the corresponding goals. Where such measures are implemented using dedicated system functionality, corresponding functional dependability requirements should be specified and assigned to specific elements of the system architecture. Functional requirements are implementation-independent specification, and can be later refined into technical requirements, considering specific design and implementation within the target system. This refinement requires that the target system either has a preliminary or mature architecture designed, so that its key subsystems and/or components have been already identified.
6. **Dependability Hardware (HW) and Software (SW) Requirement Specification.** From the technical requirements identified previously at the level of subsystems, detailed HW and SW requirements can be assigned to components of the corresponding type.
7. **Implementation.** While not explicitly part of the approach, technical implementation is expected to occur as the set of detailed component requirements is completed.

8. **HW & SW Requirements Verification.** As component implementation completes, they are verified against the corresponding requirements specified for them.
9. **Dependability Concept Verification & Validation.** Technical dependability concepts are verified as components are integrated into subsystems. Then, functional dependability concepts are verified and validated as the sets of higher-level requirements for the corresponding dependability measures are implemented.
10. **Dependability Goal Verification & Validation.** The initial dependability goals are finally verified and validated to be correctly designed, implemented, and yielding acceptable residual dependability risk across the application.

An overview of this approach can be seen in Figure 28. The V-model lifecycle is shown, flowing from top-left, starting with the System Definition stage, descending towards the Implementation stage at the centre, and ending at the top-right, at the System Validation stage. Dashed arrows represent types of arguments that can be captured within the assurance case, based on [75]. The included types are:

- **Rationale**, which are arguments capturing the reasoning with which higher-level specifications, requirements, or activities, are linked to lower-level ones.
- **Satisfaction**, which are arguments that evaluate whether the dependability requirements have been satisfied by corresponding work products. For example, satisfaction arguments can be made to establish that the dependability goals are satisfied by reviewing their validation results.
- **Means**, which are arguments that explain why the assurance activities which yielded requirements and other work product results were performed in an appropriate manner.
- **Organizational Environment** arguments address the question of whether the organization developing the target system has an appropriate culture for doing so. As an example of such requirements, Part 2 of ISO 26262 provides guidance regarding appropriate management of functional safety, and Part 5 of ISO 21434 similarly specifies guidelines for organizational cybersecurity management.

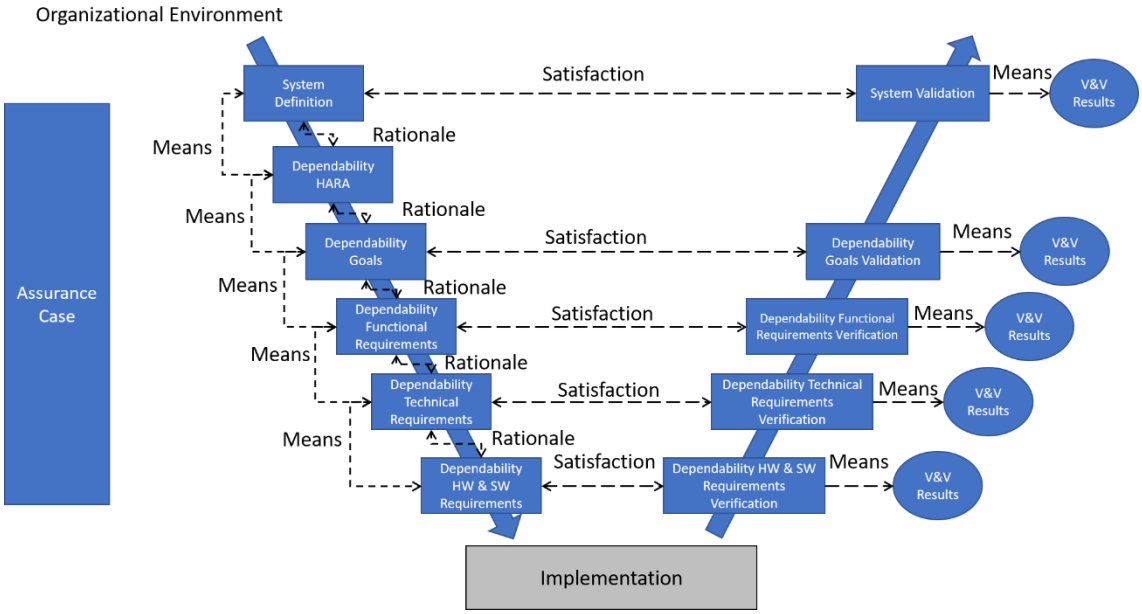


Figure 28 - Assurance Case Workflow (based on [75])

As mentioned in Section 2.3, in cases where safety (or a different dependability property) is mission-critical, the above process can prioritize performing HARA that targets that property, specify corresponding dependability goals for the identified HEs, and then

provide those goals as input for the security-focused TARA. The TARA can then simultaneously address the potential attacks against the safety/dependability properties, while still identifying security-specific issues (e.g., privacy).

4.2. BIECO Tool Support for Dependability Assurance during Development

BIECO tools can be used the above approach as follows (and correspond to the phases described in Figure 28):

- ResilBlockly (BIECO D6.2) can support the above process from the security perspective, by:
  - Modelling the system architecture, an activity consistent with the System Definition phase.
  - Identifying potential security threats against the target system, an activity consistent with the Dependability HARA (TARA) phase.
  - Identifying specific Weaknesses or Vulnerabilities of the target system, an activity which can be used to derive relevant security requirements.
- safeTbox[9] can instead focus on the safety/dependability aspects, by:
  - Modelling the system architecture, an activity consistent with the System Definition phase. To avoid potential redundancy and/or duplication errors, the system model from ResilBlockly can be imported to maintain a consistent architecture.
  - Identifying potential safety/dependability HEs against the target system, corresponding to the Dependability HARA phase.
  - Specifying dependability goals to protect against the identified HEs, as per the corresponding phase of the workflow.
  - Modelling the assurance case, using the GSN notation.
  - Qualitative and quantitative analysis of fault trees, which enables investigation of sources of dependability-related failure, specification of corresponding requirements, and verification of said requirements.

To understand the relationship of the tools in BIECO, we reproduce the overview of the BIECO tool workflow from D2.3 (p. 31) in Figure 29.

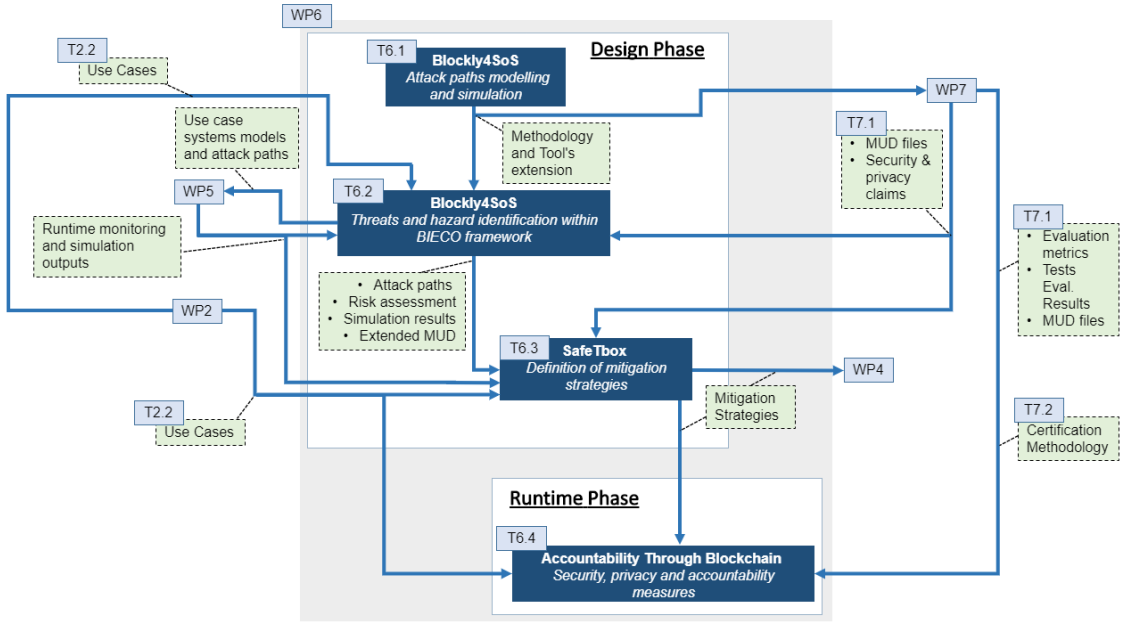


Figure 29 - BIECO Tool Workflow Overview

As indicated in the figure, information from ResilBlockly regarding e.g., security threats, attack paths, simulation results, and extended MUD files processed by the tool, can be propagated from the tool to safeTbox. For details regarding the use of ResilBlockly, the

[9] https://safetbox.de/

reader is referred to BIECO D6.2, its user's manual. More details on the information import/export across tools is provided in Section 6.1.

Once the information is imported into safeTbox, safeTbox can be used to extend the existing model and complete the assurance case. Section 7 contains a basic example of using safeTbox to model the artifacts mentioned in section's 4.1 workflow.

5. ConSerts Methodology for Dependability Risk Mitigation

In this section, the methodological aspects of our approach towards mitigating risk with ConSerts will be discussed. We begin with an overview of how the method extends the engineering workflow based on the guidelines from related standards, as outlined in sections 2.1 and 2.2. We then discuss how ConSerts can be tailored for systems employing ICTs. Finally, we explain how ConSerts acquire evidence for reconfiguration, via generic monitor integration, but also specific opportunities for integration with BIECO's resilience concepts from WP4, and the extended MUD files from WP7.

5.1. Extending ConSert Creation for Safety-Security

ConSerts view the system from the perspective of a Service-Oriented Architecture (SOA). In this sense, the emphasis is on the relationship between interrelationships across systems, rather than the structure of the systems themselves.

Therefore, the ConSert workflow for a given system consists of:

1. **Provided Services Specification.** Specifying the set of services provided by the system.
2. **Service Contract Specification**. For each provided service, a service contract specifies the set of supplied services which are required by the system to provide the given service.
3. **Service Dependability Concept**. ConSerts are predefined modular certificates, and their certification refers to both their functional and their non-functional aspects, the latter notably including dependability properties as well. At this step, the dependability concept phase, which is part of the system development lifecycle (seen in Figure 28, Section 4.1), can be extended. To deliver its provided service while managing the associated dependability risks, a clear understanding of the measures (and corresponding requirements) in place to control those risks is needed. The ConSerts approach builds upon this understanding in the following steps.
4. **Variability Analysis.** Dependability concepts are often defined according to worst-case assumptions regarding the operational situation. This limits the flexibility of the system's adaptation. To address systems in dynamic conditions, and effectively adapt to them, the variability of operational situations must be considered, in combination with the adaptation capabilities of the system itself. Such adaptation options can be, for example, considered from the situation analysis executed during development (e.g., as part of HARA). The important distinction here is the extension of the analysis scope beyond the worst-case situations. Examples of how such analyses can be applied for the automotive domain can be found in [34], [76], and [77].
5. **Contract Endpoint Specification.** Based on the contract and the combinations of operational situations and service adaptation capabilities, the required set of demands from external services, and RunTime Evidence (RtE) associated with each provided service can be specified.

As an example of how this can be modelled, Figure 30 shows a ConSert featuring a provided service ("Grid Outage Detection"), which can be offered at 3 levels of guarantee ("Normal", "Degraded Availability", and no guarantee). The service imposes demands on a required service, "AMI Grid Information", for "Normal" and "Low" availability accordingly.

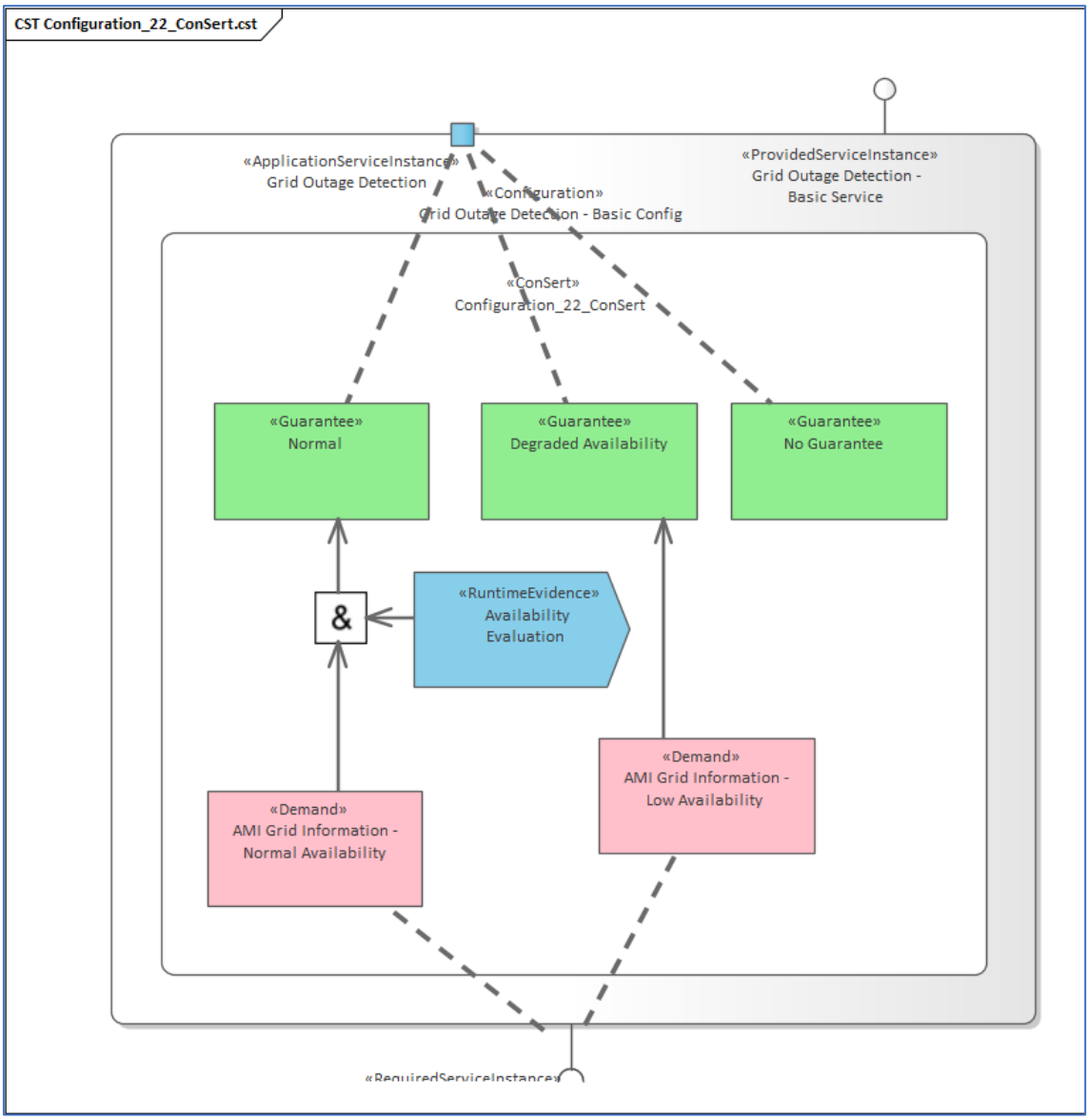


Figure 30 - Example of simplified ICT Gateway ConSert

5.2. Integration of Intrusion Tolerance Countermeasures

When system architectures feature Intrusion Tolerant Countermeasures, e.g., N-Version Programming (NVP) (sections 2.8 and 3 provide detailed background and analysis on available options), the corresponding ConSert could take advantage of the degree of consensus reached across the variants to estimate the level of confidence with respect to the provided service.

An example of how this can be depicted can be seen in Figure 31, where the ConSert shown previously has been slightly adapted. It now features two RtEs which evaluate whether all variants have achieved consensus regarding the response of the Grid Outage Detection service, or only the minimal majority. In the former case, the system can provide the service normally, with confidence that unanimous agreement across the variants is unlikely to hide potential intrusions. In the latter case, it is possible that intrusion has occurred, therefore a more conservative service guarantee could be provided in that case. The choice of these guarantees depends on the service dependability concept established during development, as per the workflow described in Section 5.

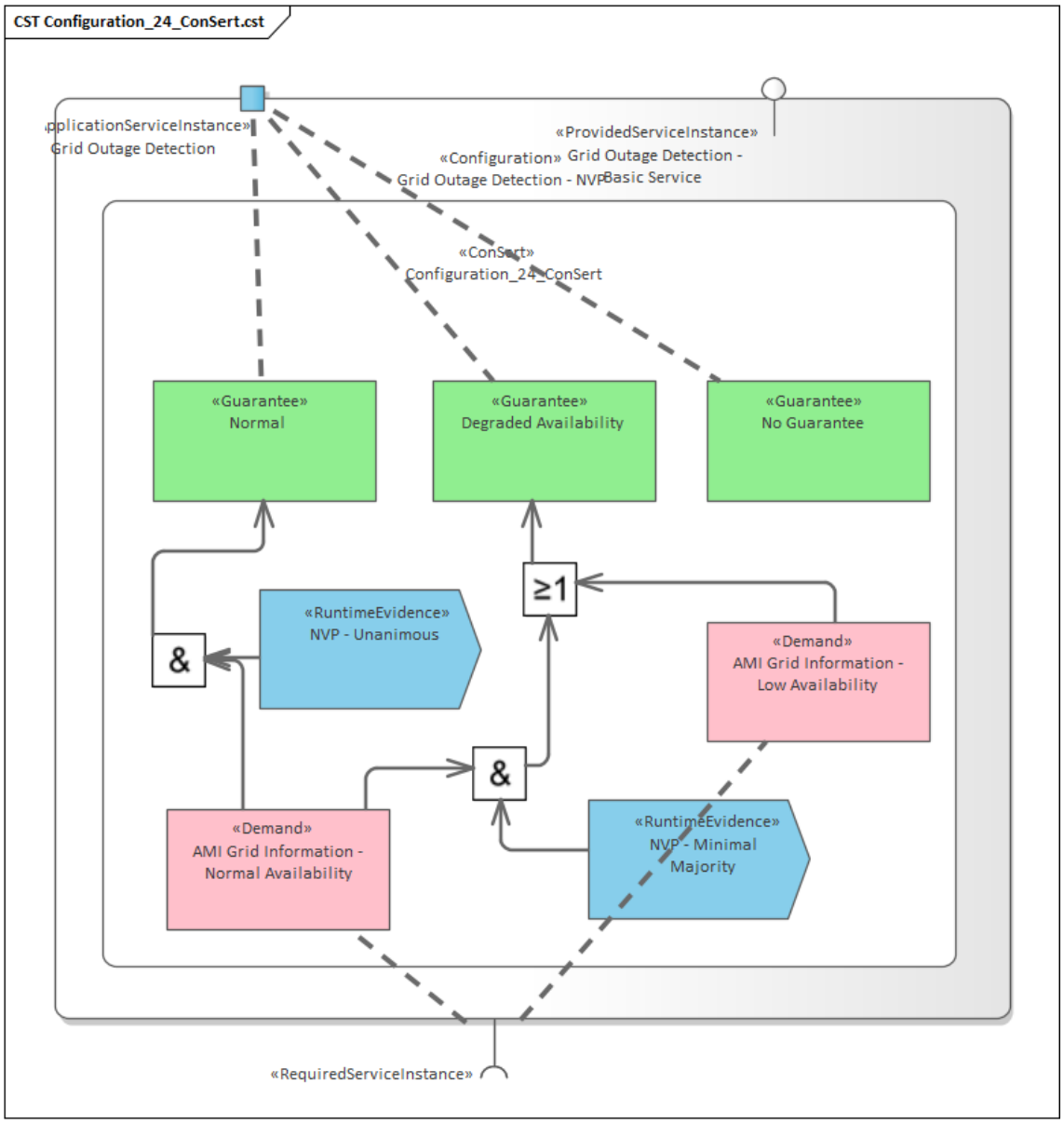


Figure 31 - Example of simplified ICT Gateway ConSert w/ NVP

5.3. Incorporating Monitoring Evidence

The RtEs found within ConSerts represent monitoring of relevant local conditions by the host system. However, ConSerts RtE monitors are not typically interested in directly monitoring nominal perception information. Instead, an RtE monitor focuses on yielding evidence in favor (or against) the integrity of services or information relied-upon by the provided service.

For example, if we consider a robot navigating through a physical environment, its visual perception could provide it with an estimation of nearby objects. A corresponding ConSerts RtE monitor would not directly evaluate the presence of nearby objects, but instead focus on evaluating whether the response of perception sensor is reliable or not.

In BIECO, WP4 is responsible with predicting system failures, and in particular, the method of using predictive simulation to anticipate the behavior of systems under the control of a software smart agent received as a black box, whose internals are not known. Execution of simulation models in a predictive simulated environment (BIECO WP5) can feed evidence of trust to the monitoring components. In case the predictive simulation outputs a trusted behavior of a component, then the trusted behavior signature is passed to the conformity monitoring part of the Auditing Framework (BIECO WP5), that evaluates the level of conformity between the trusted behavior execution in a simulated environment and the real-world execution. In case the predictive simulation detects a hazardous situation, a triggering for the system's internal re-configuration will enable the system to reach a safe state and assure its resilience. This work could be used as RtE for ConSerts and will be explored further as part of the planned work of BIECO in T4.3.

5.4. Hardened MUD File as a mitigation measure

One of the potential mitigation measures to carry out when a system or service cannot provide the demanded guarantee is the configuration of a stricter or hardened MUD File for the involved component.

As discussed in previous deliverables, the Manufacturer Usage Description (MUD) standard describes the network behaviour profile recommended by the manufacturer to properly function. This profile establishes a set of policies to take in account in order to limit the threat surface on a device and its connections. Section 5 of BIECO D6.1 goes deeper into the MUD standard and the MUD model.

The idea behind this mitigation strategy is to provide an alternative MUD File to the original one defined by the manufacturer when required service conditions are not guaranteed. If under more restrictive conditions it is possible to offer this service, a stricter MUD File will be deployed so that service will work with several network limitations compared to the original MUD File of the involved components. Specifically, these limitations are focused on the traffic coming from/to the device, number of communications, services allowed to access from the device, required algorithms for cryptography, authentication mechanisms, application protocols to be used in the communications and limit the accessible resources. More specific information on the possible security configuration characteristics that could be limited can be found in BIECO D6.2.

Consequently, incorporating these restrictive measures benefits the security context of the requested service in two ways. First, these measures reduce the risk of attacks when offering a service that does not comply with the security conditions, and secondly, they increase control over the established connections and over which entities, components or devices these connections are made.

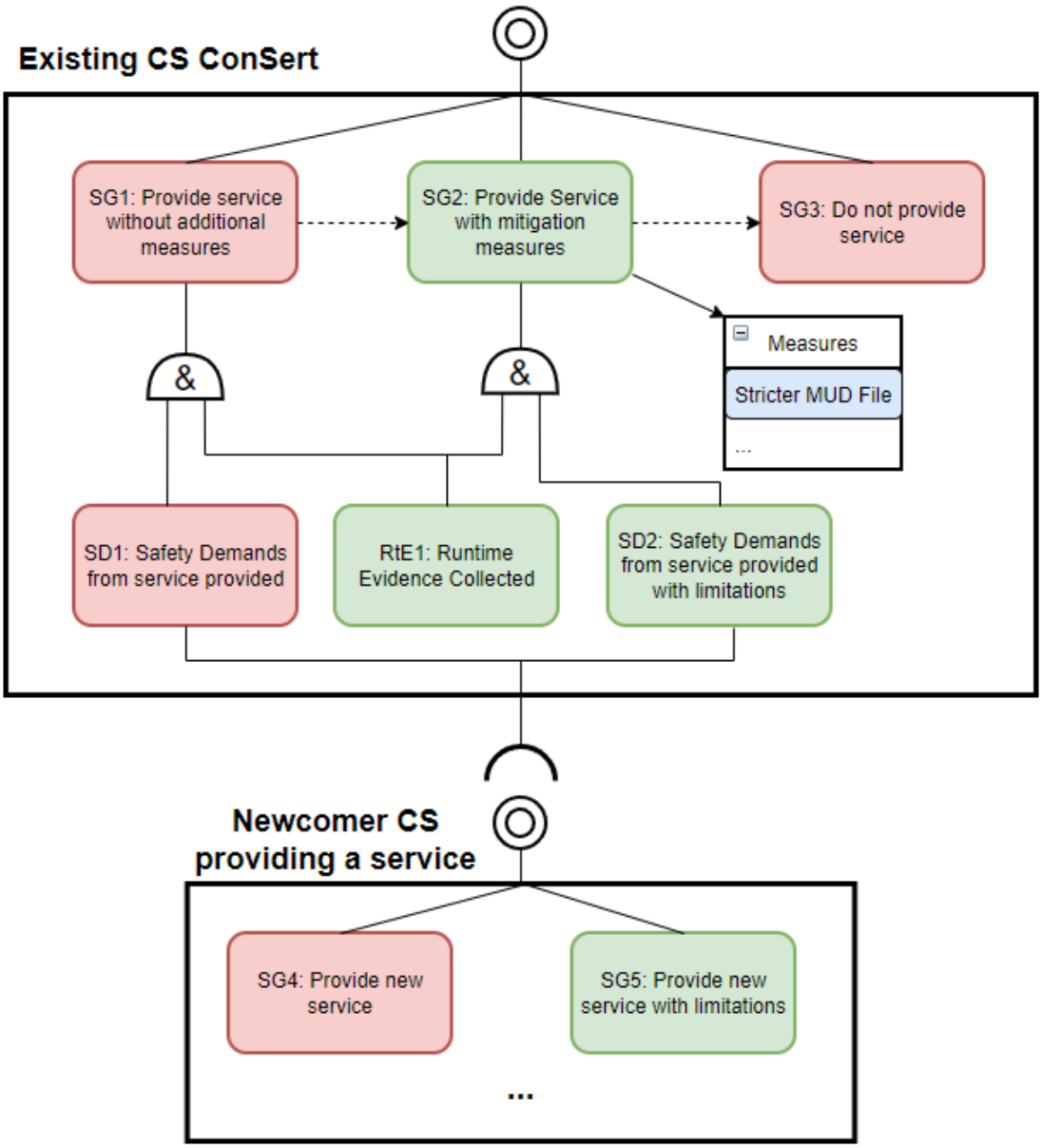


Figure 32 - Example diagram about a ConSert offering a service with a stricter MUD file

Figure 32 shows a conceptual application example of this potential mitigation measure. In this case, a newcomer CS is providing a service that is going to be used by an existing CS. The upper ConSert should consider the conditions from the bottom ConSert in order to confirm the possibility of offering the safety guarantees of his services. Since only "SG5" is able to be provided with safety guarantees, the upper service offered by the existing CS is "SG2" (provided with mitigation measures such as stricter MUD File). This decision flow is marked by the green boxes.

6. BIECO ConSert Modelling

In this section, we focus on the modelling of ConSerts in BIECO. Specifically, we begin by mapping BIECO concepts from which ConSerts shall draw upon. Then, we present the modelling approach for constructing ConSerts in the safeTbox tool. Finally, we discuss how ConSerts can be exported for later use as models, whose usage can include generating corresponding software components that implement the reconfiguration logic for the system the components are integrated with.

6.1. Mapping BIECO Artifacts to DDIs

Figure 33 provides an overview of the 2nd version of the Open Dependability Exchange (ODE) metamodel [40]. The ODE consists of the Structured Assurance Case Metamodel (SACM), which is highlighted in purple in the figure, and the remainder, referred to as the ODE Product Packages, highlighted in green. We should stress that we are not attempting to provide a detailed discussion of the ODE and its elements here; such a description can be found in [40], and a detailed specification of the ODE can also be found in its open source repository[10].

The ODE enables integrating assurance cases via SACM (which also supports GSN models) with concrete system dependability assurance artifacts. For example, within an ODE model, i.e., a DDI, a system can be represented using the ODE::Design::System element[11]. A given System can be associated with dependability requirements, including security, and dependability analyses e.g., HARA, TARA, FTA, ATA etc. The corresponding ODE::Dependability and ODE::FailureLogic metamodelling packages contain metamodelling elements for capturing such aspects.

As a DDI is compiled with increasing information during development, it can become valuable as a medium for synchronizing assurance activities. This can be especially the case across interdisciplinary teams e.g., safety and security, where shared terminology and overlapping methods may pose coordination challenges.

The use of a common model such as the DDI can also support interoperability between tools, as less effort needs to be spent producing pairwise-specific import/export mechanisms for each tool or invent custom formats for ad-hoc interoperation. In this regard, the common tool adapter developed to support interoperability and automate DDI-related activities is particularly suited for tool exchange. Further details are provided in Section 7, Figure 40.

[10] https://github.com/Digital-Dependability-Identities/ODE
[11] The notation is interpreted as the metamodeling element named System from the metamodeling package named Design of the ODE profile

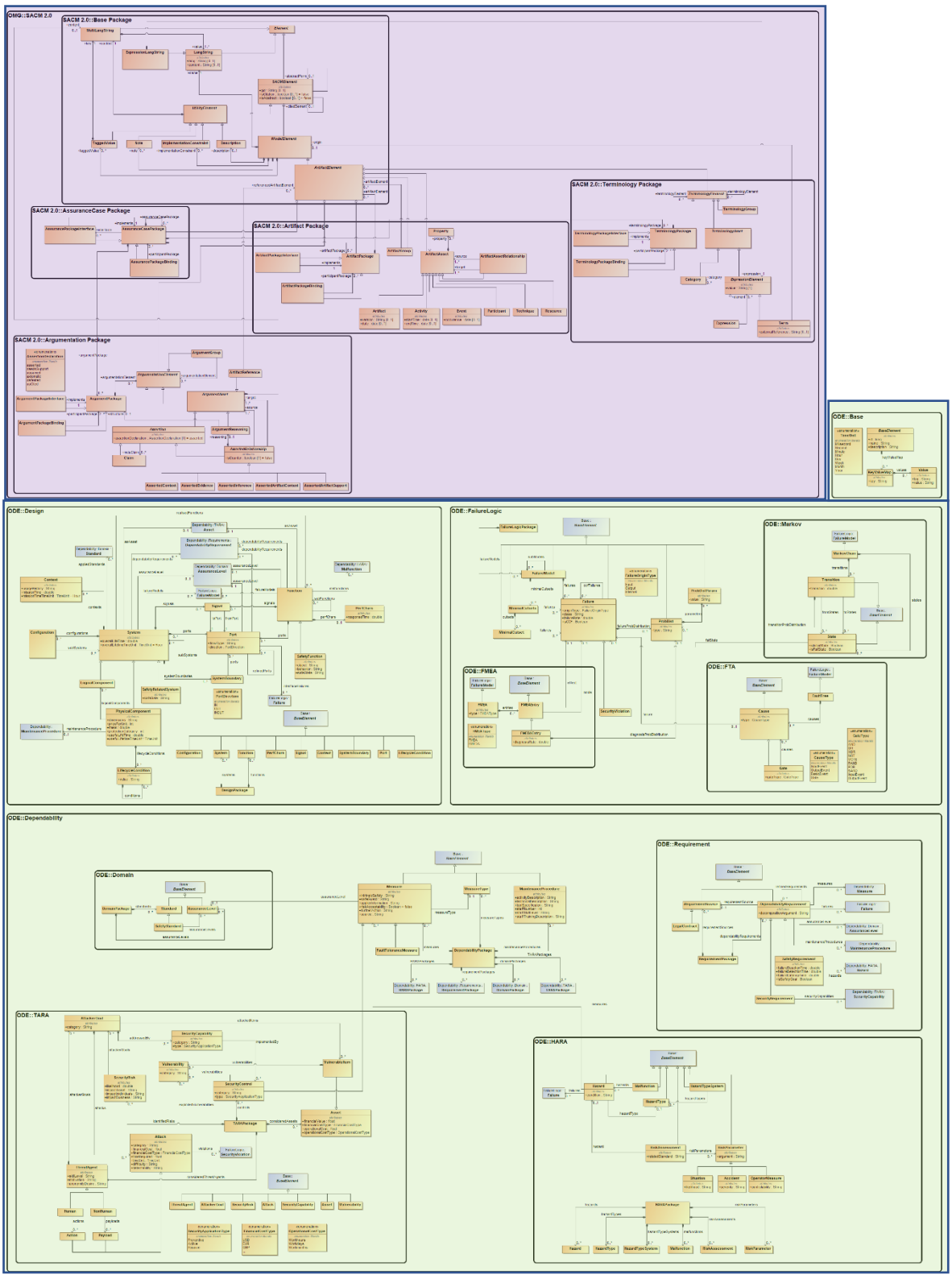


Figure 33 - Overview of the ODE v2 Metamodel

6.2. Modelling BIECO ConSerts using safeTbox

To produce ConSerts models, such as those seen in Figure 30 and Figure 31, a prototype extension of safeTbox has been developed by Fraunhofer IESE. SafeTbox itself is an 'add-in' extension of the modelling tool Enterprise Architect (EA). The prototype extension extends the built-in EA UML profile to introduce new modelling elements for ConSerts e.g., ConSerts diagrams, Guarantees, Demands, etc.

Figure 34 shows the options the user has to model elements on each ConSert diagram. Guarantees and Demands have already been introduced in Section 5. Invariants can be included in a ConSert diagram to represent preconditions that must be valid for the ConSert to be valid. Invariants can be checked at runtime, and the host application can then determine how to address the situation e.g., falling back to a fail-safe state, switching to a back-up ConSert, and/or informing the user, if possible.

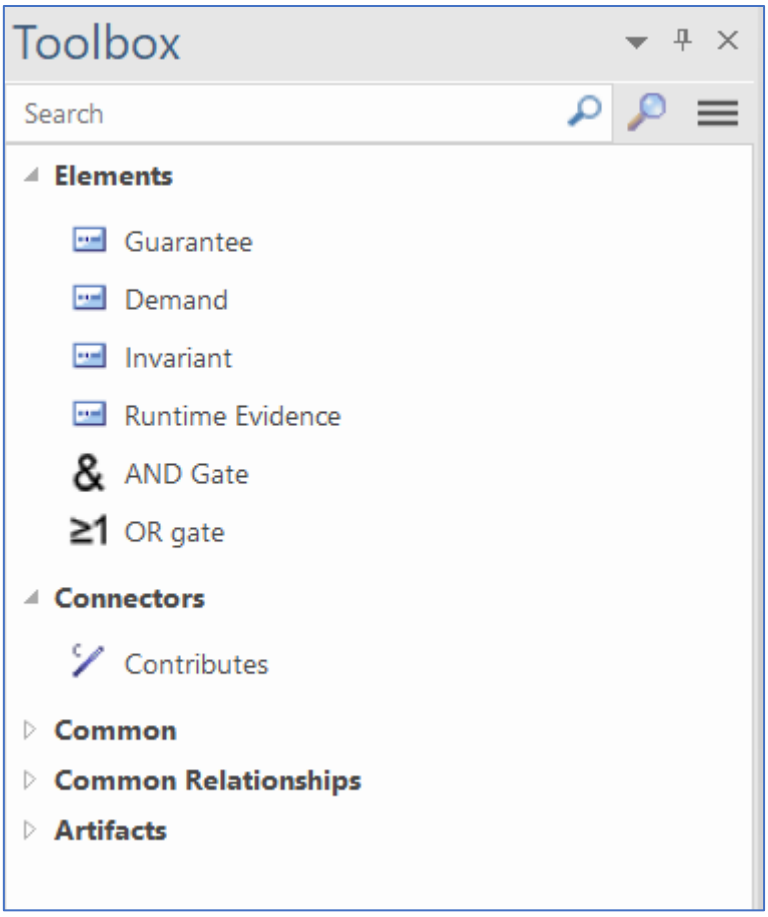


Figure 34 - Enterprise Architect Custom safeTbox Toolbox for ConSerts

6.3. Generating Deployable ConSerts via conserts-rs

In [78], the conserts-rs command-line tool is introduced. The tool can be used to parse XML representations of ConSerts models, such as the one seen in Figure 35, and generate source code that can be integrated into a variety of platforms. The process for converting from models to code can be seen in Figure 36, and a sample of the code generated from the XML example is shown in Figure 37.

```
<?xml version="1.0" encoding="ASCII"?>
<allInOne:System xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:allInOne="allInOnePkg" xsi:schemaLocation=
"allInOnePkg ../metamodel/allInOneMetaModel.ecore" globalId="42" name="TBA" description="">
  <configurations name="configuration" description="" configurationRanking="1">
    <conSert name="consert" description="">
        <guarantees name="SG4" description="">
            <safetyProperties name="" description="">
                <refinement name="Near Environment Unoccupied by Humans" description=""/>
            </safetyProperties>
        </guarantees>
        <guarantees name="SG5" description="">
            <safetyProperties name="" description="">
                <refinement name="Far Environment Unoccupied by Humans" description=""/>
            </safetyProperties>
        </guarantees>
        <runtimeEvidence name="RtE3" description="Near Zone Unoccupied."/>
        <runtimeEvidence name="RtE4" description="Approximation Speed of Detected Object &lt;=
        1m&#x2f;s"/>
        <runtimeEvidence name="RtE5" description="Installation Approved by Health and Safety
        Engineer"/>
        <runtimeEvidence name="RtE6" description="Far Zone Unoccupied."/>
        <runtimeEvidence name="RtE7" description="Approximation Speed of Detected Object &lt;=
        2m&#x2f;s"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@runtimeEvidence.0"
        targetElement="//@configurations.0/@conSert/@gates.0"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@runtimeEvidence.1"
        targetElement="//@configurations.0/@conSert/@gates.0"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@gates.0" targetElement=
        "//@configurations.0/@conSert/@gates.1"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@runtimeEvidence.2"
        targetElement="//@configurations.0/@conSert/@gates.1"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@gates.1" targetElement=
        "//@configurations.0/@conSert/@guarantees.0"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@runtimeEvidence.3"
        targetElement="//@configurations.0/@conSert/@gates.2"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@runtimeEvidence.4"
        targetElement="//@configurations.0/@conSert/@gates.2"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@gates.2" targetElement=
        "//@configurations.0/@conSert/@gates.3"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@runtimeEvidence.2"
        targetElement="//@configurations.0/@conSert/@gates.3"/>
        <guaranteePropagation sourceElement="//@configurations.0/@conSert/@gates.3" targetElement=
        "//@configurations.0/@conSert/@guarantees.1"/>
        <gates name="Or_0" gateType="Or"/>
        <gates name="And_1" gateType="And"/>
        <gates name="Or_2" gateType="Or"/>
        <gates name="And_3" gateType="And"/>
        </conSert>
  </configurations>
</allInOne:System>
```

Figure 35 - Example ConSerts XML (Ecore) file

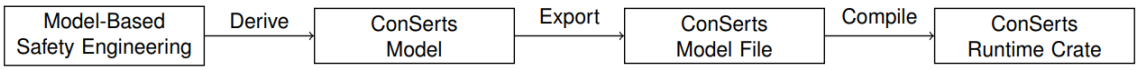


Figure 36 - ConSerts Model to Runtime Code Conversion Process

```
use consert_edcc2021::{properties::*, *};

#[app(device = target, peripherals = true)]
const APP: () = {
  struct Resources {
    safe: bool,
    rtp: properties::RuntimeProperties,
    monitor: monitor::Monitor,
    // ...
  }
  // ...
  #[task(resources = [safe, monitor, rtp])]
  fn evaluate_safety(
    cx: evaluate_safety::Context) {

    let resources = cx.resources;
    // Move current sample to monitor
    resources.monitor.add_sample(
      *resources.rtp);
    *resources.rtp = RuntimeProperties::unknown();

    // Evaluate safety
    let rte = resources.monitor.get_sample();
      *resources.safe =
        guarantees::SG4::evaluate(&rte);
  }
}
```

Figure 37 - Sample of generated ConSerts code from XML

The tool is written in the Rust[12] programming language, which enables (among other features), memory-safe and minimal-overhead code. Although the tool also outputs Rust code, it can flexibly target many platforms, including:

- Robot Operating System (ROS)[13]
- Embedded Systems via the Real-Time Interrupt Concurrency Framework[14]
- C/C++ via Foreign Function Interface[15]

The generated code still needs to be instrumented with the host platform, which is platform-specific. For instance, the generated code can be executed as a ROS node and interact with the existing application through the ROS publish/subscribe topic mechanism. Instead, integration into a C++ would involve invoking host application code to instrument the ConSerts RtEs.

[12] https://www.rust-lang.org/
[13] https://www.ros.org/
[14] https://rtic.rs/0.5/book/en/
[15] https://doc.rust-lang.org/nomicon/ffi.html

7. Exemplary Application on ICT Gateway

In this section, the approach is illustrated by applying it on a simplified model of the BIECO ICT Gateway use case (see Net2DG project, D1.3[16]).

Figure 38 shows how the ODE profile[17] (the metamodel of DDIs) can be seen in ResilBlockly's profile designer after being imported. Using the ODE profile, models that are very close to DDIs can be exported, which facilitates the import process into tools supporting DDIs, such as safeTbox.

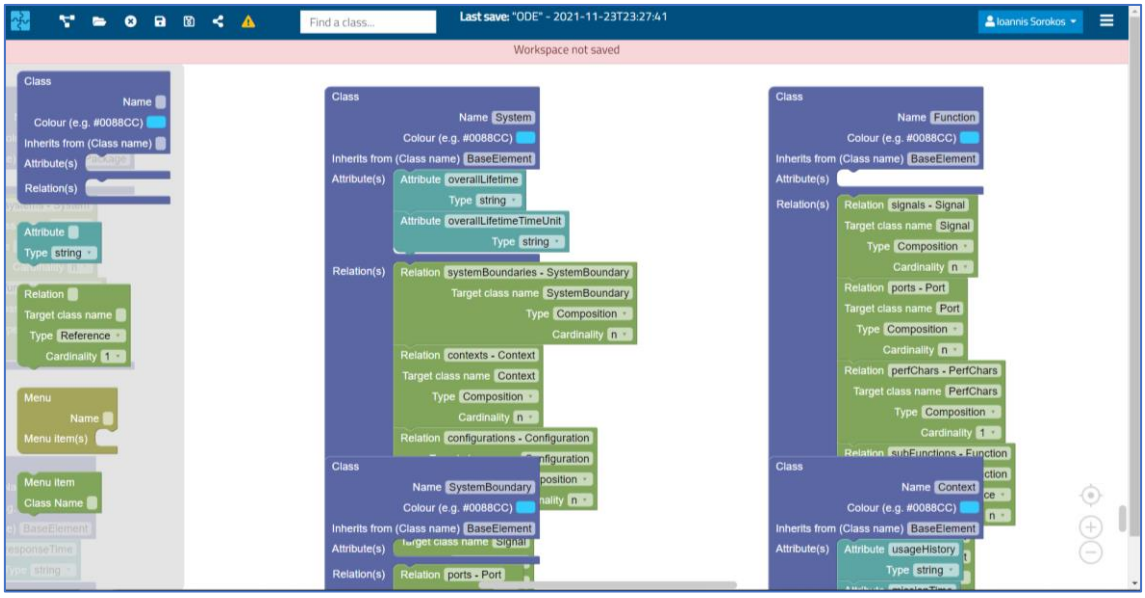


Figure 38 - ODE Profile in ResilBlockly

Using the profile, a similar model (being an instance of the ODE metamodel) can be created to depict the subject system i.e., the ICT Gateway. Once complete, a security risk analysis can also be performed, to identify corresponding weaknesses and vulnerabilities that might threaten the system. For instance, Figure 39 shows 2 weaknesses having been specified for part of the system under development, and their corresponding risk evaluated as 'High' and 'Moderate'.

2 weaknesses related to the Block ""

Weakness ID	Weakness Title	Weakness Description	Common Consequences	Likelihood of Exploit	Details	Severity	Likelihood	Risk
CWE-400	Uncontrolled Resource Consumption	The software does not properly control the allocation and maintenance of a limited resource, thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources.		High		10	Mod	High
CWE-770	Allocation of Resources Without Limits or Throttling	The software allocates a reusable resource or group of resources on behalf of an actor without imposing any restrictions on the size or number of resources that can be allocated, in violation of the intended security policy for that actor.		High		10	Low	Moderate

Figure 39 - Security Risk Analysis in ResilBlockly

The model exchange process can be seen in Figure 41. Step 3 is currently required due to minor incompatibility in the exported format, and involves applying ODE-specific types to the generically-exported ones from ResilBlockly. Step 4 can be executed using the 'common tool adapter' developed as part of the DEIS project [79]. An overview of the adapter can be seen in Figure 40. The adapter allows any tool which supports file,

[16] http://www.net2dg.eu/wafx_res/Files/Net2DG_D1.3_30.08.2019_with%20disclaimer.pdf
[17] https://github.com/Digital-Dependability-Identities/ODE

network, or remote procedure call interoperation (via the Apache Thrift framework [18]) to transform to/from DDIs, and execute scripts in the Epsilon language[19] on the provided DDIs. The typical usage of the adapter is for generating DDI files, using the ODE profile [20] specified in the Eclipse Modelling Framework (EMF) [80].

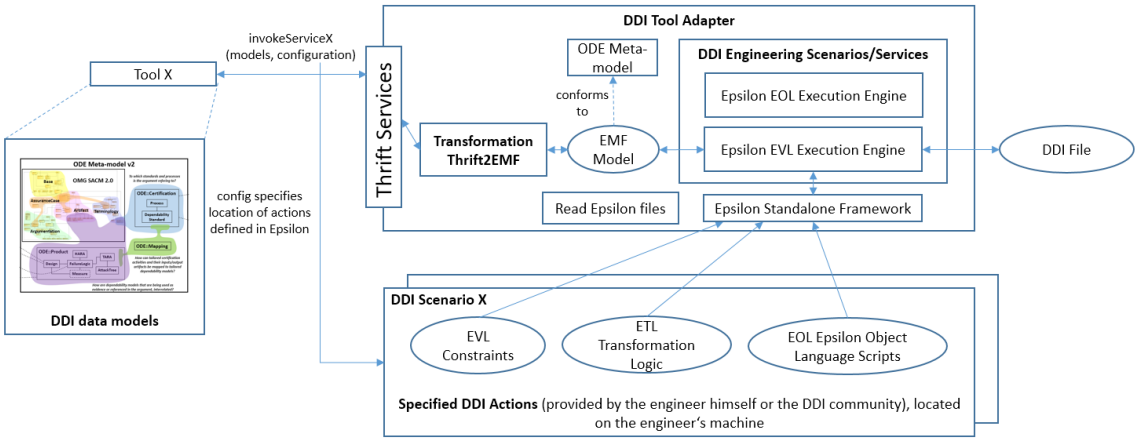


Figure 40 - DDI Tool Adapter (from [41])

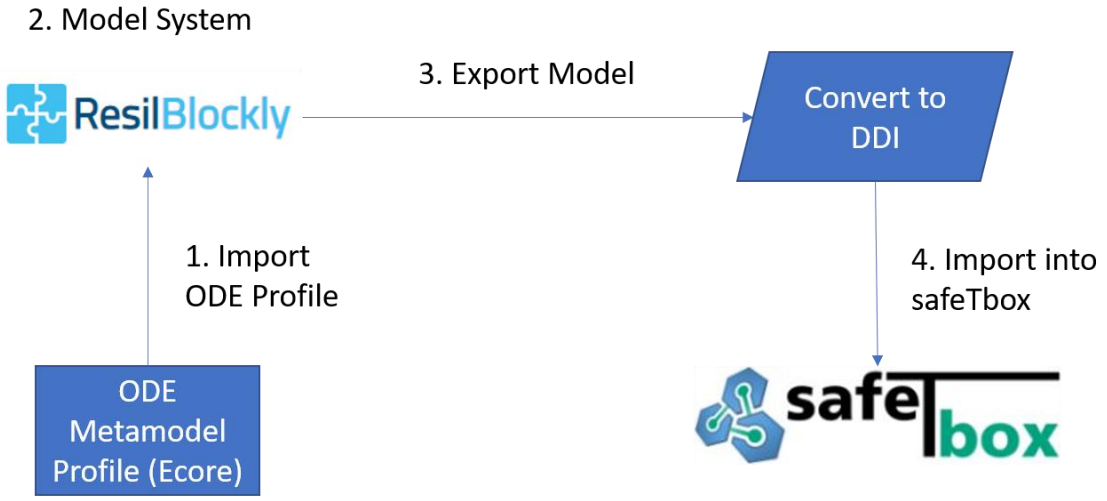


Figure 41 - Model Exchange Overview

The imported model can be further tailored and expanded upon in safeTbox. This is depicted in Figure 42, where the Smart Grid Production System (top left) produces power for the Consumer(s) (top right). The Distribution Service Operator (DSO) organization transfers the power to their consumers through the Distribution System (i.e., the smart grid). A given operator working for the DSO organization monitors the activity of the grid through information propagated by the ICT Gateway. The ICT Gateway collects grid information through the Internet. Depending on the information observed, the operator can decide to control the production system and/or the distribution system to avoid power outage. The specific functionality of interest is the monitoring and detection of power outage in the grid.

[18] https://thrift.apache.org/
[19] https://www.eclipse.org/epsilon/doc/eol/
[20] https://github.com/Digital-Dependability-Identities/ODE

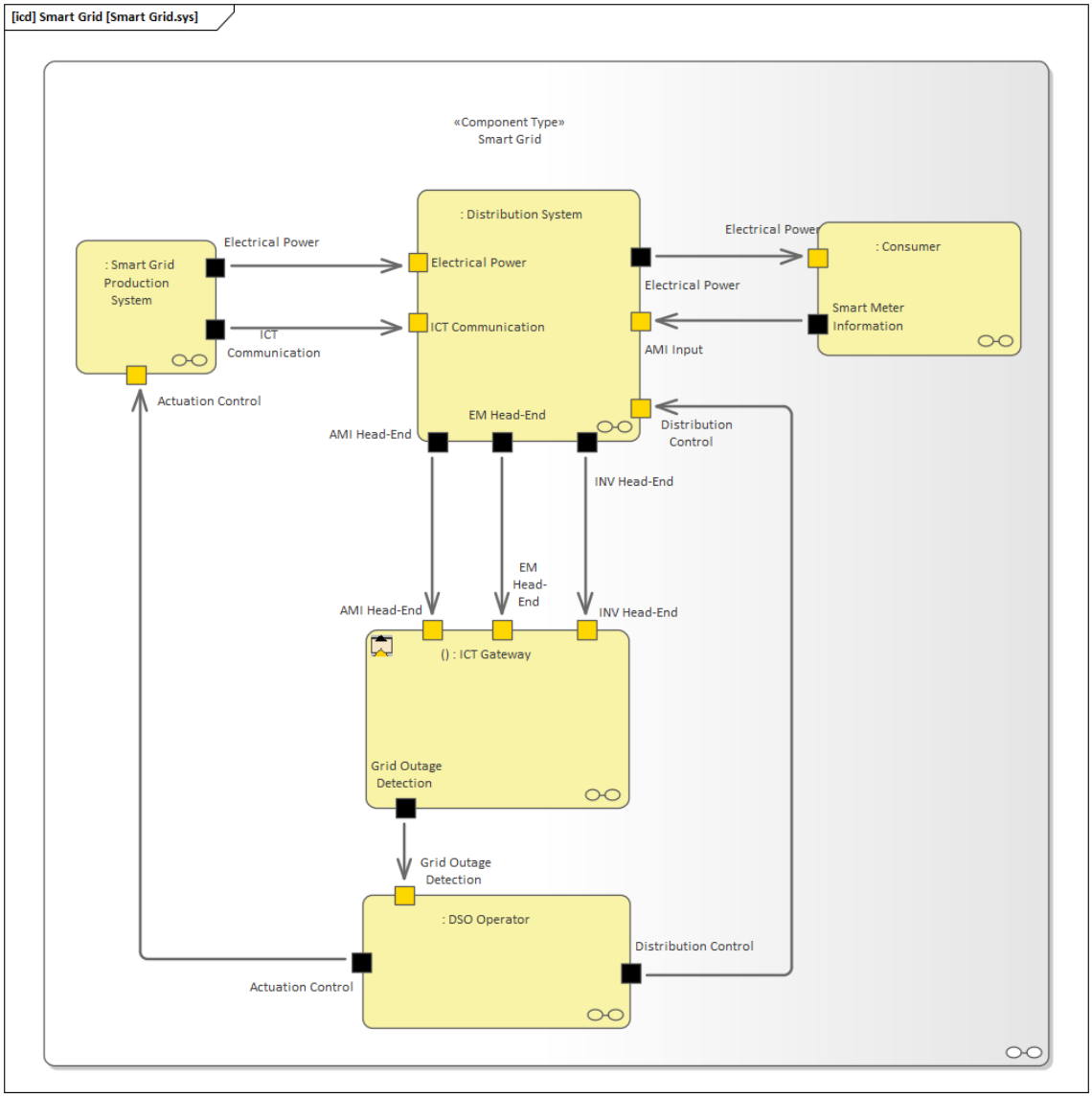


Figure 42 - Simplified model of ICT Gateway use case Smart Grid

As our scope for the model is very simple, Figure 43 describes the ICT Gateway simplifying its interface to include only the Safety & Resilience component. In contrast, a more detailed model would also consider the other components, as listed in Figure 25.

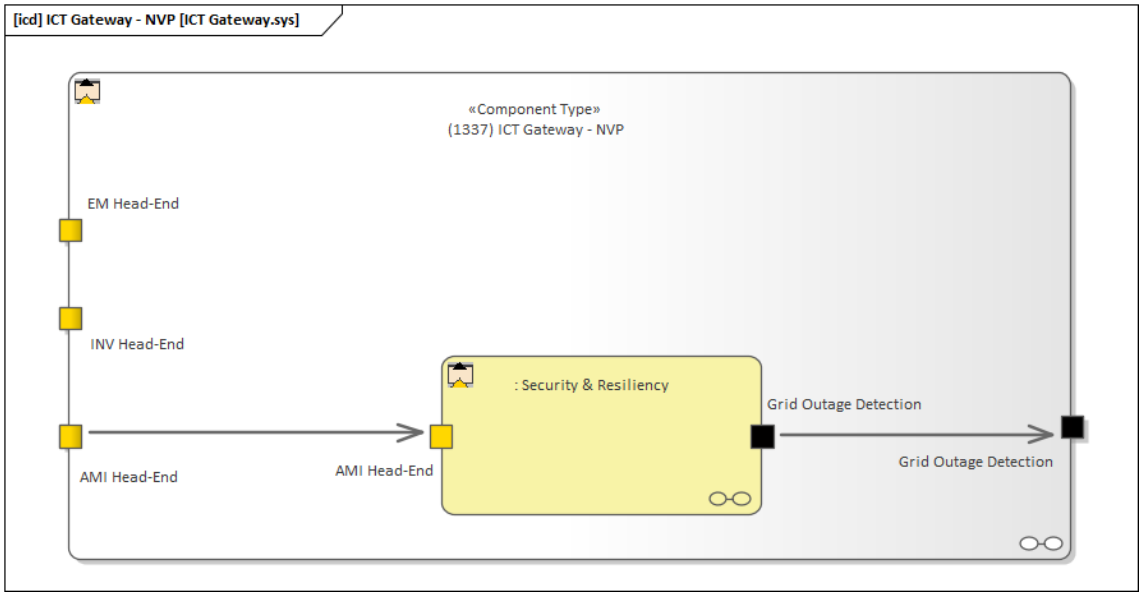


Figure 43 - ICT Gateway, simplified internal view

Based on the preliminary model, an initial (dependability) HARA can be performed. safeTbox follows a spreadsheet approach for the HARA, as seen in Figure 44. The spreadsheet seen captures functions used, which in this case is the “Grid Outage Detection” of the gateway.

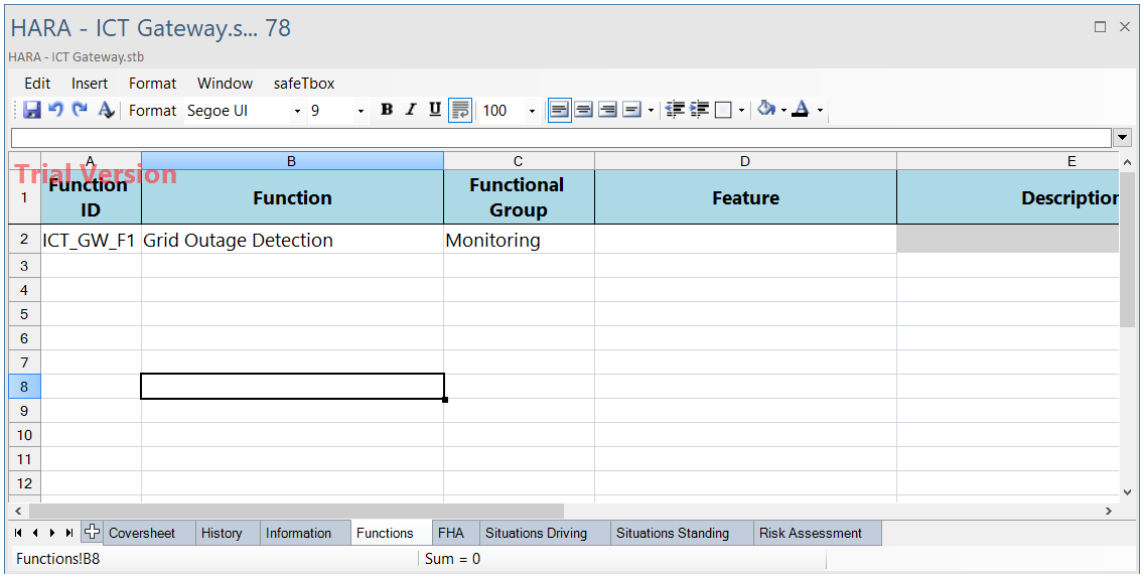


Figure 44 - ICT Gateway HARA - Function Sheet

In the Functional Hazard Analysis (FHA) sheet, seen in Figure 45, the individual failure modes with which the function can fail are distinguished, the malfunctions that arise from each case are specified, and the potential system-level effects of the failures are also recorded. Each entry can be mapped (if relevant) to one or more hazards i.e., events with negative impact on the application, resulting from the associated malfunctions.

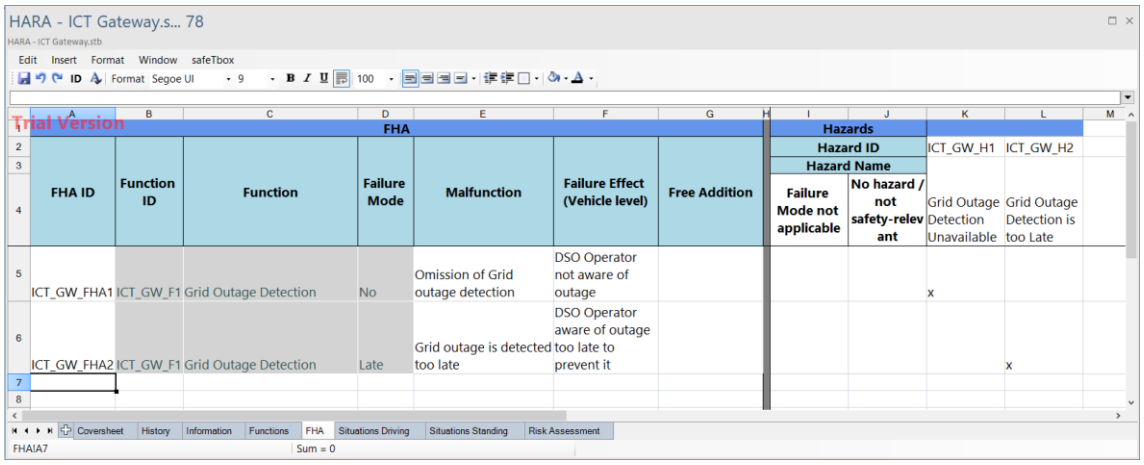


Figure 45 - ICT Gateway HARA - FHA Sheet

The final HARA sheet is the Risk Assessment sheet, whose overview is seen in Figure 46. The sheet's rows correspond to the Hazardous Events (HEs) (combinations of Hazards and Operational Situations), which are evaluated in terms of specific risks. This is seen in more detail in Figure 47, where the risk factors related to the Grid Outage Detection HEs are evaluated. Once an HE risk has been evaluated, corresponding goals can be set to protect against it, as seen in Figure 48.

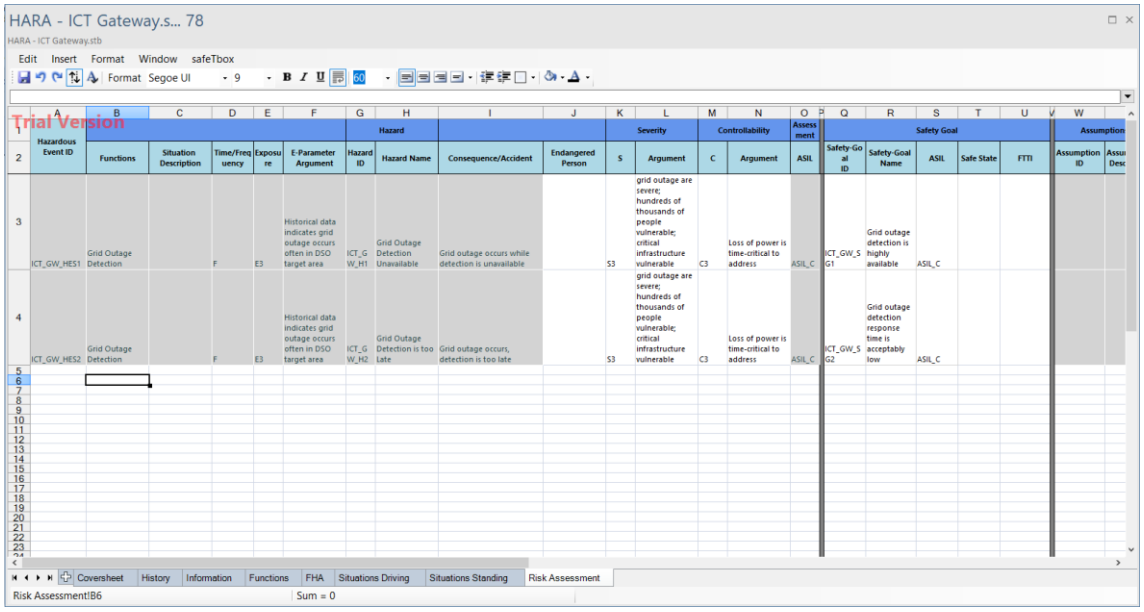
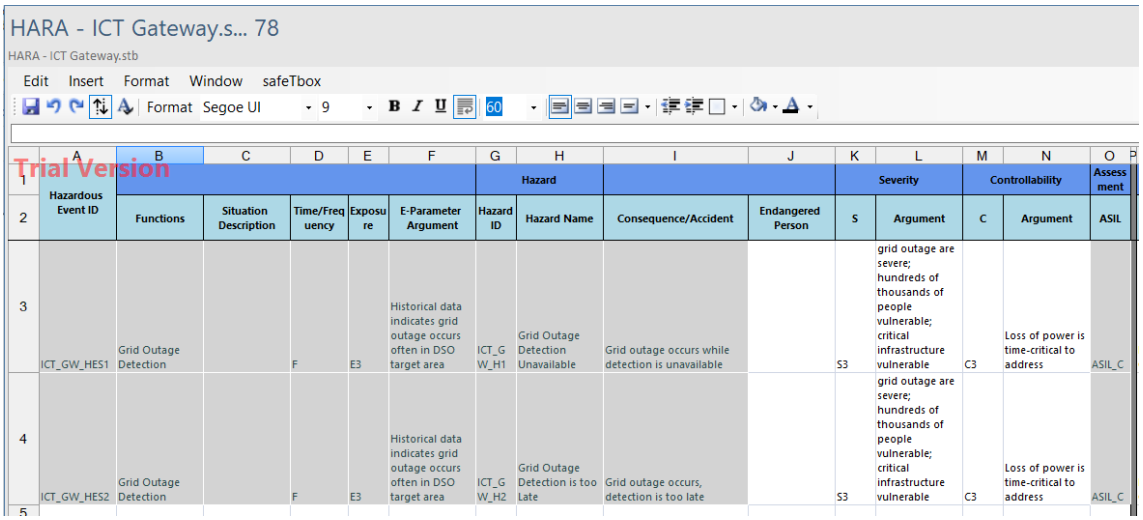
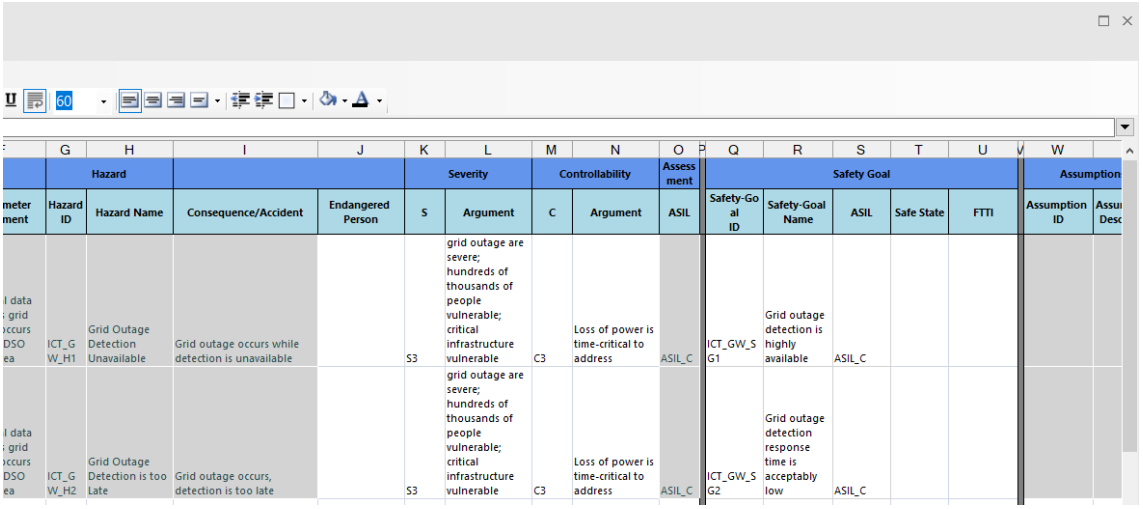


Figure 46 - ICT Gateway HARA - Risk Assessment Sheet



Hazardous Event ID	Functions	Situation Description	Time/Frequency	Exposure	E-Parameter Argument	Hazard ID	Hazard Name	Consequence/Accident	Endangered Person	S	Argument	C	Argument	ASIL
						Hazard				Severity		Controllability		Assessment
ICT_GW_HES1	Grid Outage Detection		F	E3	Historical data indicates grid outage occurs often in DSO target area	ICT_GW_H1	Grid Outage Detection Unavailable	Grid outage occurs while detection is unavailable		S3	grid outage are severe; hundreds of thousands of people vulnerable; critical infrastructure vulnerable	C3	Loss of power is time-critical to address	ASIL_C
ICT_GW_HES2	Grid Outage Detection		F	E3	Historical data indicates grid outage occurs often in DSO target area	ICT_GW_H2	Grid Outage Detection is too Late	Grid outage occurs, detection is too late		S3	grid outage are severe; hundreds of thousands of people vulnerable; critical infrastructure vulnerable	C3	Loss of power is time-critical to address	ASIL_C

Figure 47 - ICT Gateway - Risk Assessment - Hazardous Event Assessment



Hazard ID	Hazard Name	Consequence/Accident	Endangered Person	S	Argument	C	Argument	ASIL	Safety-Goal ID	Safety-Goal Name	ASIL	Safe State	FTTI	Assumption ID
				Severity		Controllability		Assessment	Safety Goal					Assumption
ICT_GW_H1	Grid Outage Detection Unavailable	Grid outage occurs while detection is unavailable		S3	grid outage are severe; hundreds of thousands of people vulnerable; critical infrastructure vulnerable	C3	Loss of power is time-critical to address	ASIL_C	ICT_GW_SG1	Grid outage detection is highly available	ASIL_C			
ICT_GW_H2	Grid Outage Detection is too Late	Grid outage occurs, detection is too late		S3	grid outage are severe; hundreds of thousands of people vulnerable; critical infrastructure vulnerable	C3	Loss of power is time-critical to address	ASIL_C	ICT_GW_SG2	Grid outage detection response time is acceptably low	ASIL_C			

Figure 48 - ICT Gateway HARA - Risk Assessment - Safety Goal Specification

Once the set of dependability goals have been specified, they need to be refined into detailed requirements. Towards this end, iterative cause analysis can be used e.g., via (Component) Fault Tree (CFT) Analysis, seen in Figure 49. In the figure, the triangle elements in black indicate output failure modes, whereas yellow triangle elements indicate input failure modes. Squares link failure modes to a given component's ports, thereby linking the architectural diagram, e.g., seen in Figure 42, and its causal failure logic. CFTs are hierarchical, as depicted by the "Safety & Resiliency" sub-CFT, whose details are encapsulated in the diagram of Figure 50. The latter figure also features 2 basic events i.e., fundamental causes that could trigger system/component failure. The basic events are linked via a Boolean logic OR gate, therefore either could trigger the system failure i.e., "Grid Outage Detection Omission/Late". The green analysis results shown in Figure 49 capture this logic, and detailed results of the analysis can be reviewed in Figure 51.

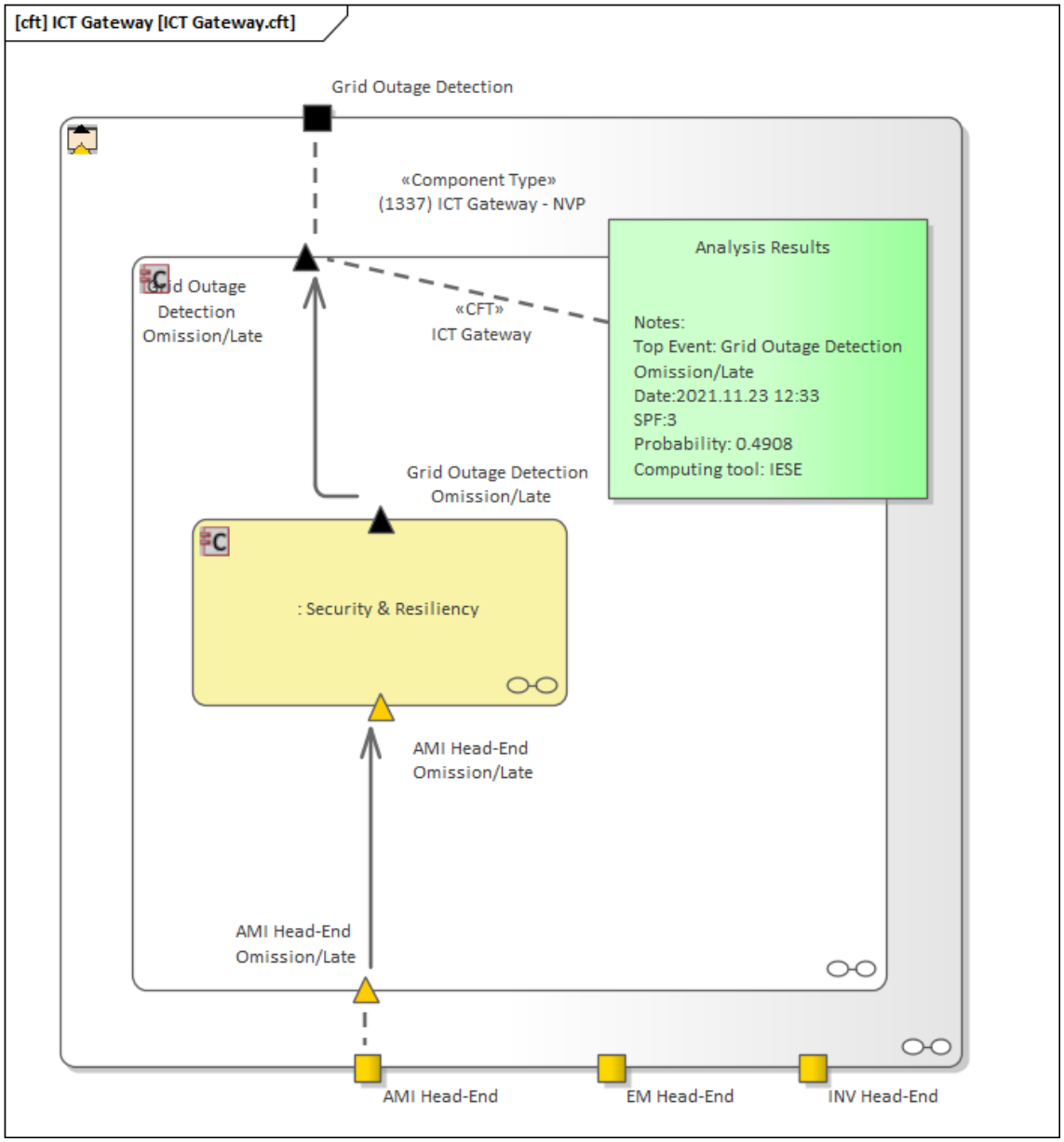


Figure 49 - ICT Gateway CFT

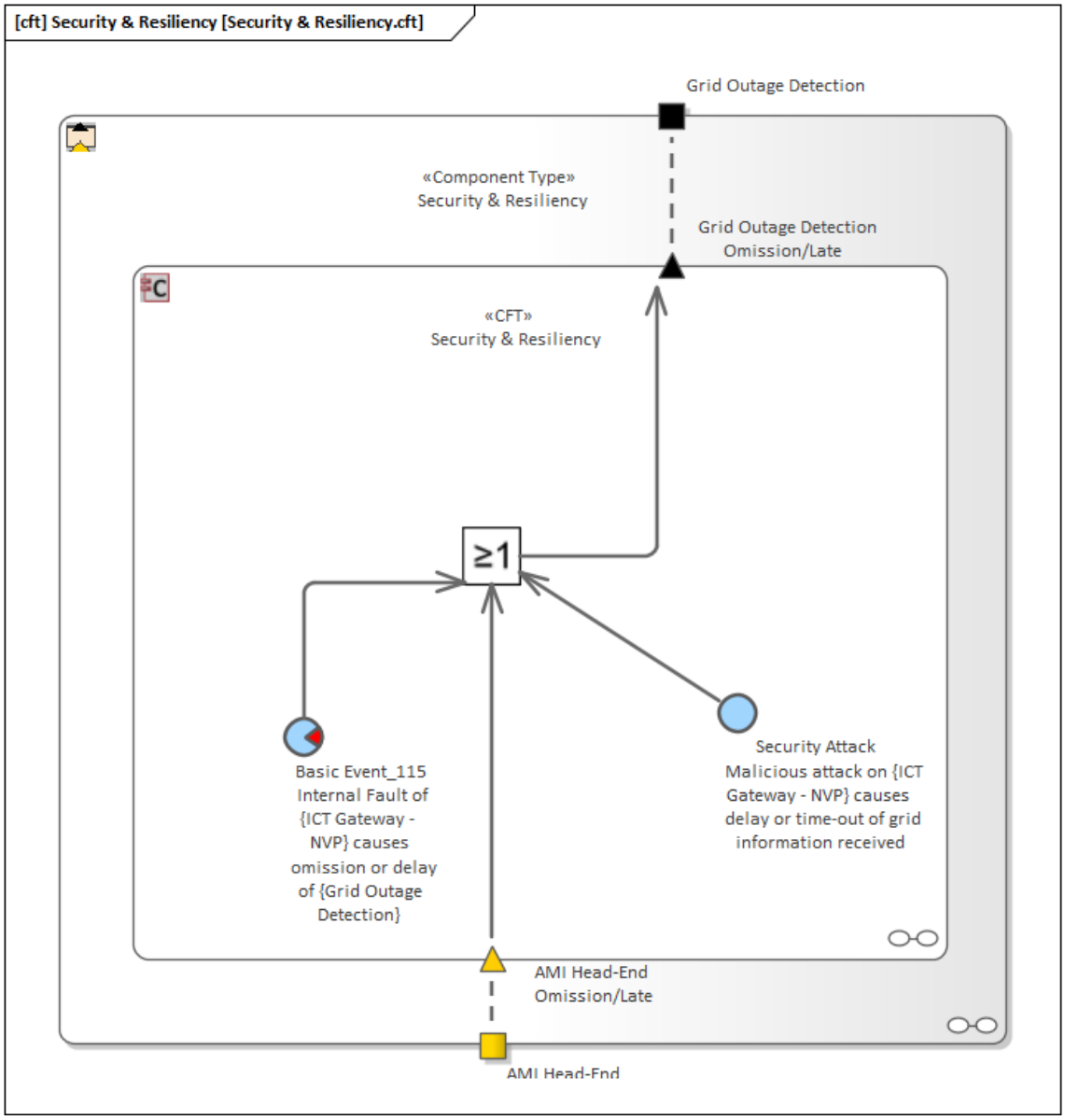


Figure 50 - Safety & Resiliency CFT

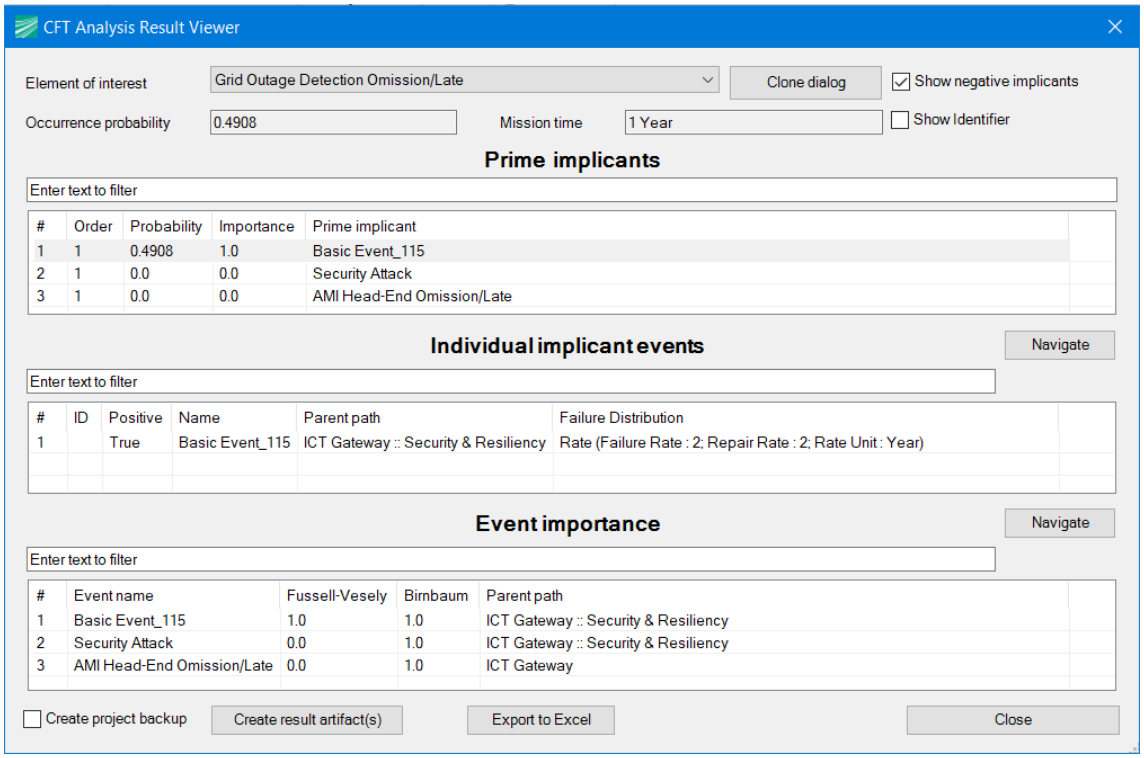


Figure 51 - ICT Gateway - CFT Analysis Results

Using the information from the previous steps taken, an assurance case can be modelled using the GSN notation, as seen in Figure 52. Note that the assurance case is similarly simplified, focusing on illustrating the tool usage for applying the approach. The Top-Goal in the figure claims that the ICT gateway is acceptably dependable. The argument is based on addressing relevant dependability properties e.g., availability (Strategy_83). This is justified given that the overall Smart Grid application is also directly affected by availability of the ICT Gateway (Justification_85). Finally, Away_Goal_95 references an external goal (Goal_84), claiming that the residual risk against availability of the ICT gateway has been shown to be acceptably low.

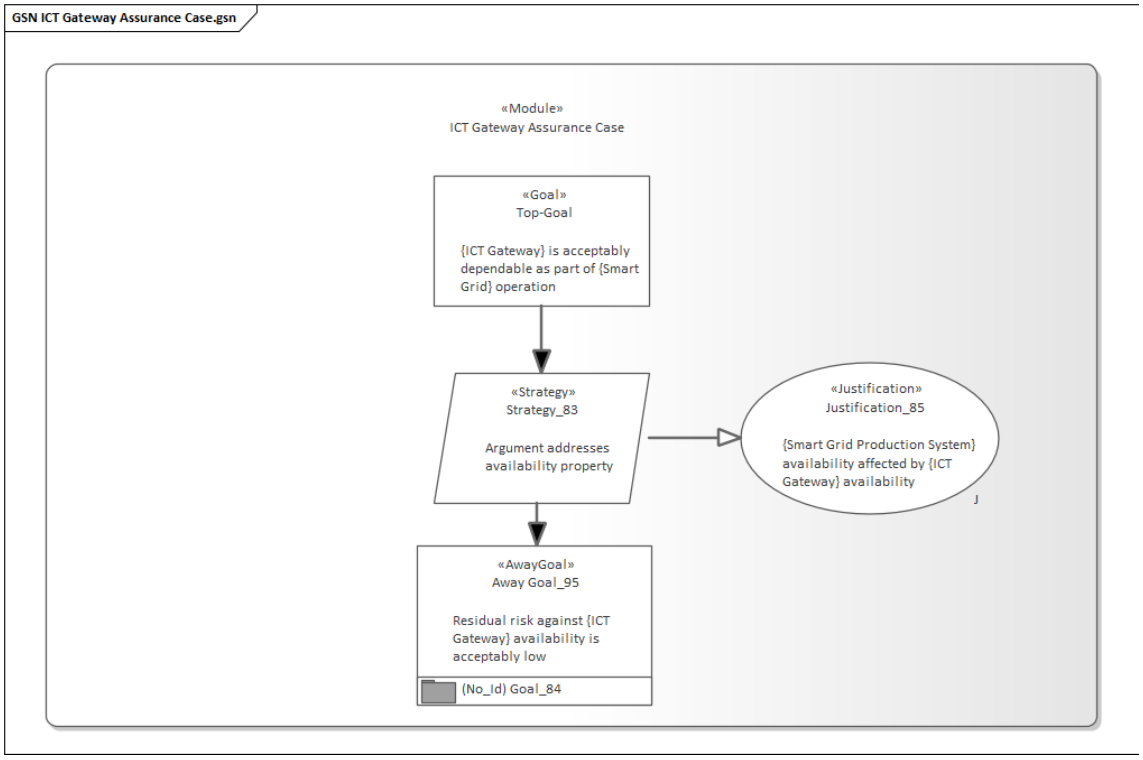


Figure 52 - ICT Gateway - Assurance Case Top Goal

Figure 53 continues down this line of argument, which addresses the two hazards identified during the HARA. The line of argument proceeds through Figure 53 to address the goal of "Grid Outage Detection" being highly available, which addresses the corresponding HE from the HARA analysis (Figure 48).

It should also be noted that the HARA, provided as context to the overall claim, is associated with an ACP that provides a claim regarding the quality of the Means (see Section 4, Figure 28) with which it was conducted (Figure 54).

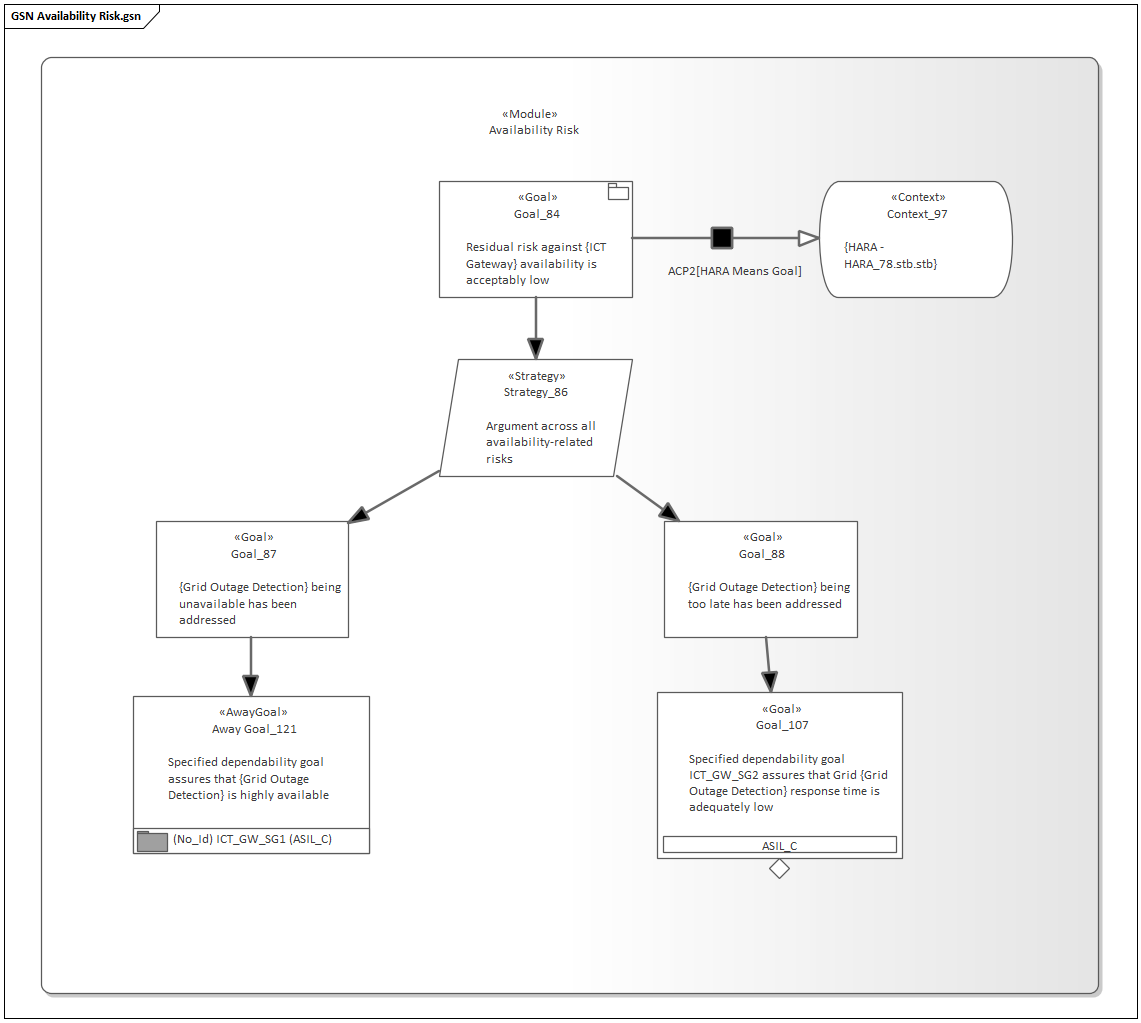


Figure 53 - ICT Gateway - Availability Risk Module

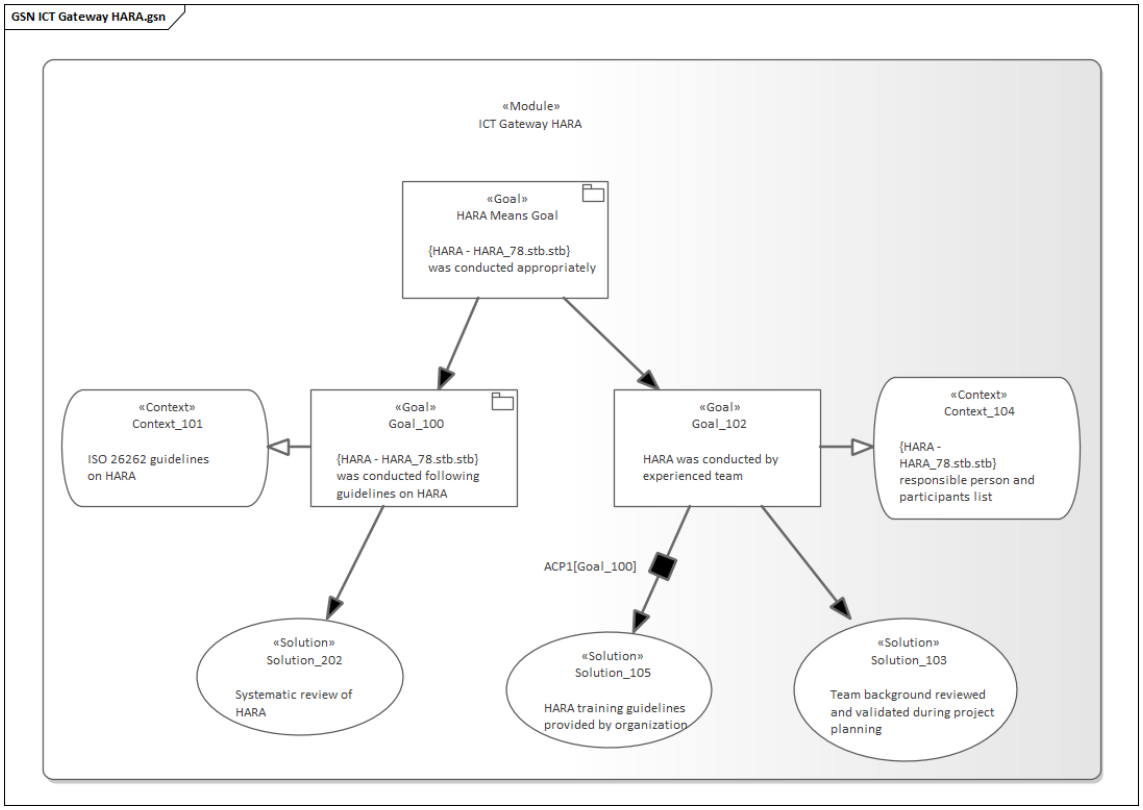


Figure 54 - ICT Gateway - HARA Module

Finally, Figure 55 argues why the Functional Dependability Concept addresses the associated dependability goal, in this case using NVP to reduce risk of service unavailability. This claim is verified by comparing the CFT analysis results before (Figure 51) and after the inclusion of the redundant variants, as seen in Figure 56 and Figure 57.

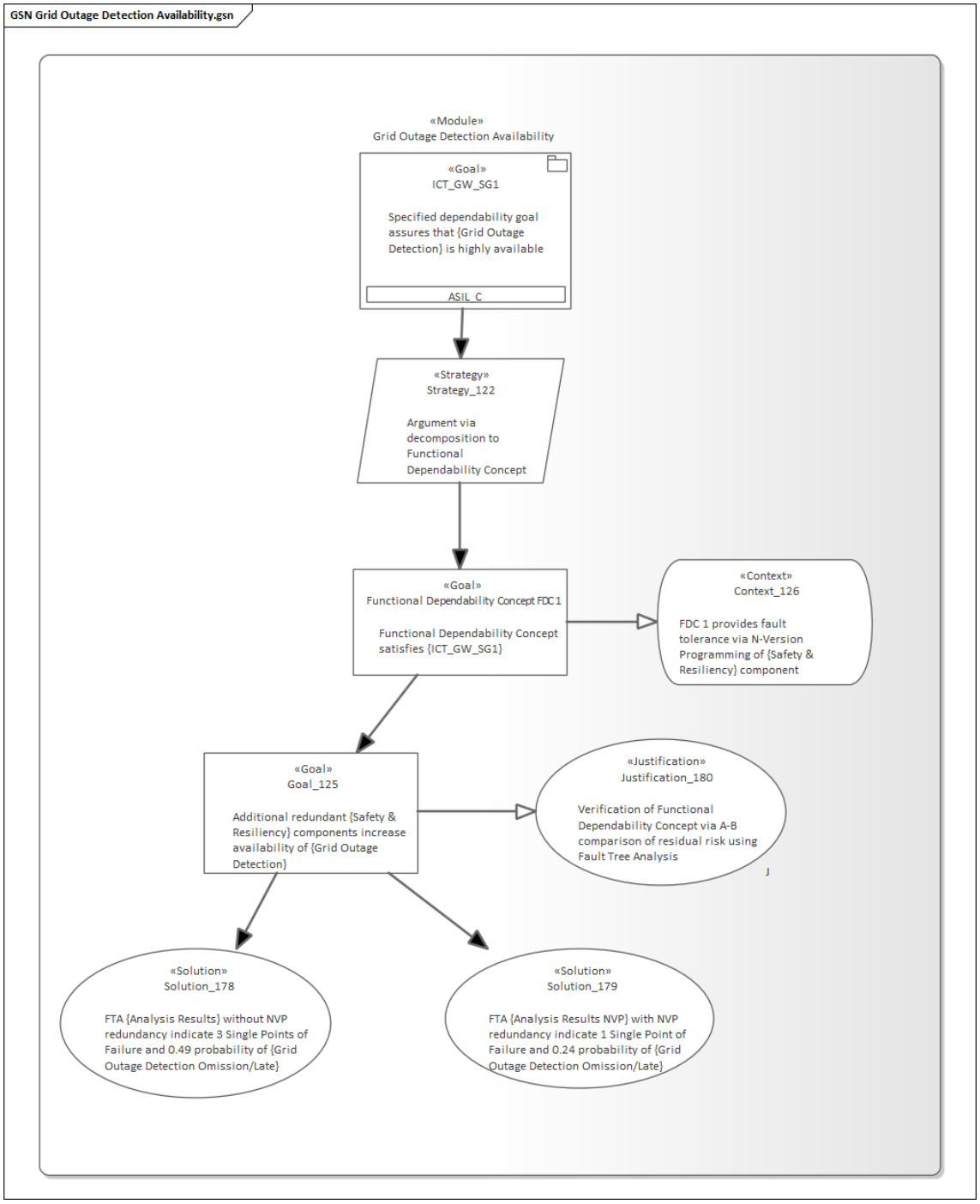


Figure 55 - ICT Gateway - Functional Dependability Concept Module

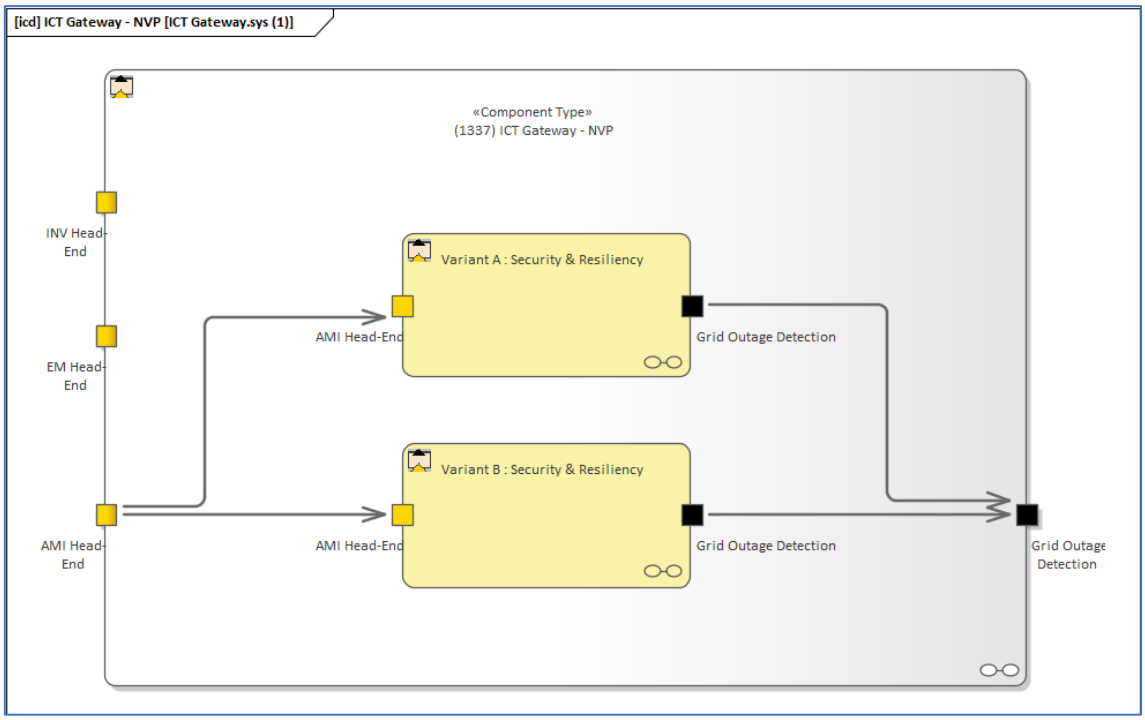


Figure 56 - ICT Gateway using 2 variants for Safety & Resiliency NVP

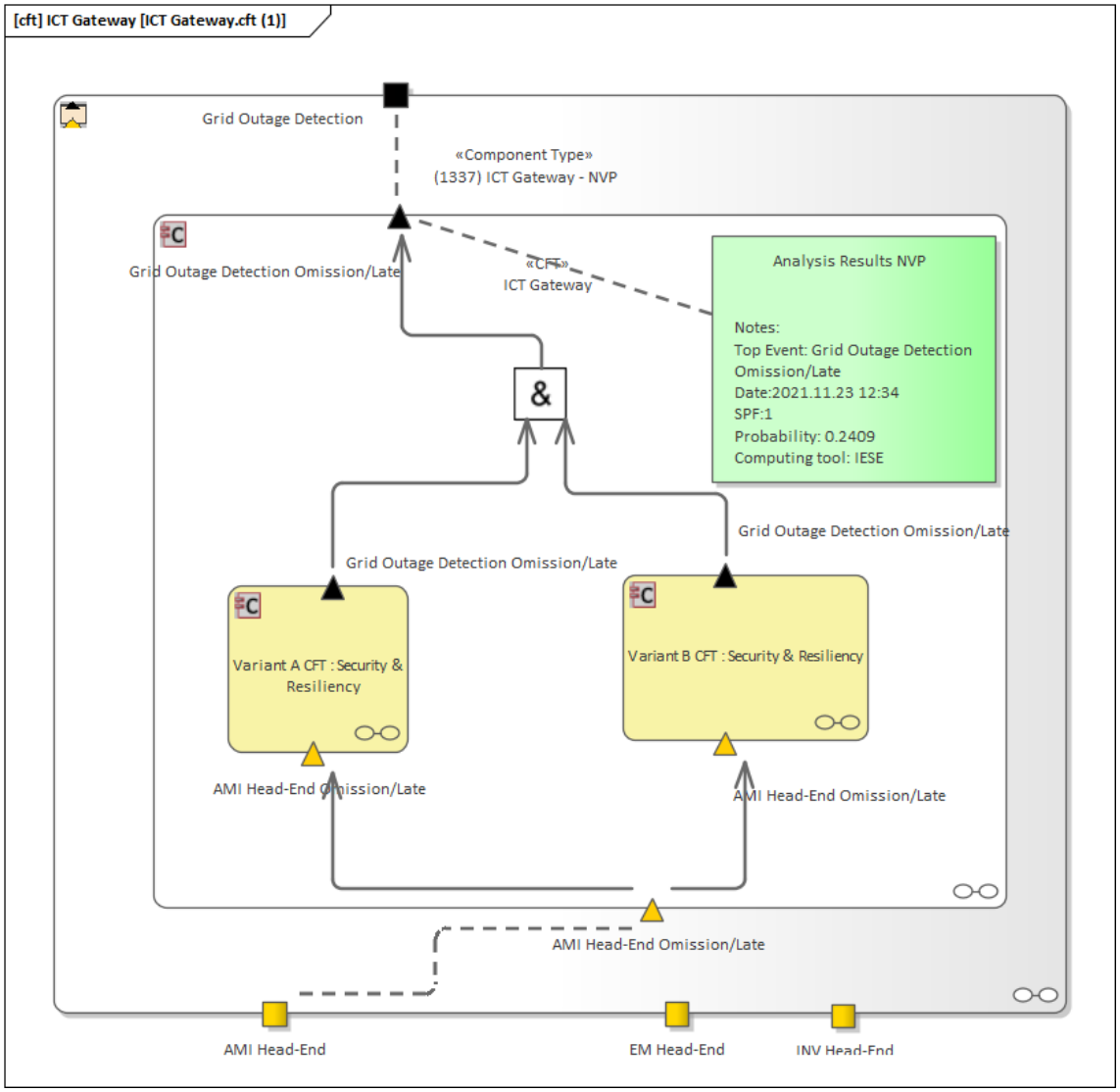


Figure 57 - ICT Gateway Fault Tree including NVP

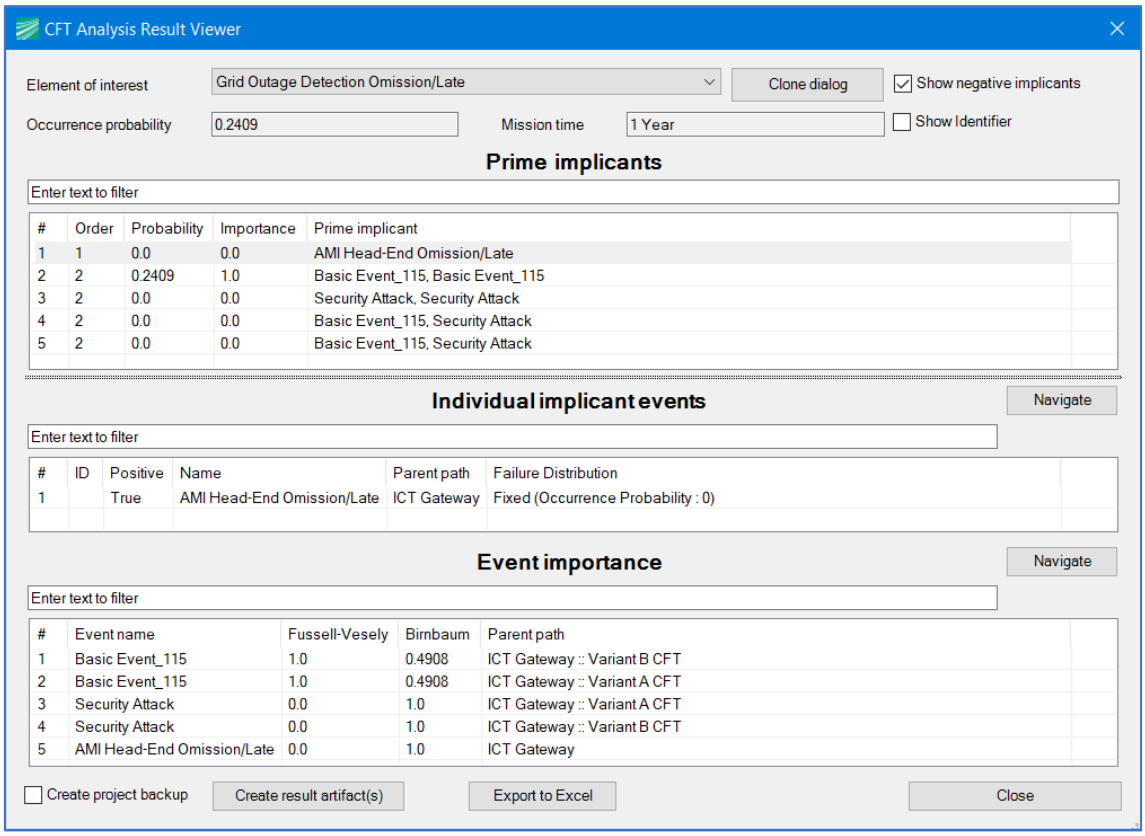


Figure 58 - ICT Gateway w/ NVP Example CFT Analysis Results

7.1. Hardened MUD File

Smart grids provide electricity to a wide range of the population. A failure in this type of system not only implies a power cut in homes, but also in highly sensitive buildings such as a hospital, where a failure in the electricity supply could have fatal consequences for the lives of many people. Ensuring its operation under certain safety conditions is crucial for supplying electricity. It is important to mention that given the importance of the continuous operation of this type of system, allowing its execution even though not all conditions can be guaranteed is a possibility that should be considered.

If the necessary conditions described in the ConSerts to ensure a dependable execution of the ICT Gateway cannot be fulfilled, the ICT Gateway could be still executed under more restrictive conditions established by the hardened MUD file. In this sense, the hardened MUD could, for example, establish a lower limit of simultaneous connections ("num-connections") per device until the conditions of the grid return to a safe mode to offer the service without restrictions.

Specifically, we present in Figure 59 a hardened MUD file for the ICT gateway, restricting the number of simultaneous connections allowed with the MQTT and HTTP protocols as well as the persistent connection time ("Keep-Alive").

On the one hand, the left part of Figure 59 contains the original MUD File configuration that would be used in a situation where the service could be offered with all security guarantees. On the other hand, the right side of Figure 59 shows the hardened MUD File used in our risk situation that forces us to offer the service with mitigation measures, in this case, a more restrictive configuration.

Figure 59 - Application of mitigation measure using a stricter MUD File

8. Summary

In this deliverable, we have provided an overview of our approach towards supporting safety and security assurance of ICT systems in terms of risk mitigation. Our approach aims to address risk originating from developmental or systematic errors (e.g., software implementation or documentation mistakes), or anticipated risk from random hardware errors, or anticipated risk from malicious actors against our system (e.g., specific security attacks).

Towards this end, we focus our approach on modelling assurance cases that can structure respective arguments of adequate risk mitigation during system development. Assurance cases are well-known in specific industrial domains e.g., automotive, and are also part of corresponding industry standards e.g., ISO 26262. Assurance cases can appropriately leverage the domain-specific analysis evidence garnered from each domain and translate the implications of the evidence in terms of overall risk. Additionally, assurance cases also enable the construction of combined safety and security risk argumentation, meaning that both aspects (and more e.g., availability) can be considered holistically, minimizing the risk of overlooking critical crosscutting concerns.

Furthermore, our approach considers how adaptive systems, operating in dynamic environments, should respond to changing conditions (including in terms of security) with regards to safety. We leverage the concept of Conditional Safety Certificates to not merely specify such adaptations (as would be the ad-hoc approach), but further guarantee that these adaptations certifiable in terms of safety.

To realize our approach, we intend to exploit and extend the above concepts in the context of BIECO's ongoing research and use cases, and we have already illustrated in this deliverable our current plan. In short, our approach:

- Provides methods appropriate for systematically structuring safety (T6.2) and security assurance claims (WP7) as part of assurance cases.
- Incorporating the risk assessment (T6.2) and security analysis (T6.1) process to provide appropriate development-time evidence of risk mitigation.
- Developing more resilient systems through the concept of Intrusion-Tolerant Architectures (WP6), intended to mask the effect of attack-induced failures, and integrate the developed redundancy schemes in the risk management process (WP6)
- Links failure and trust prediction concepts with dynamic risk management (WP4).
- Links runtime risk management and resilient adaptation (WP4).

9. Appendices

Appendix I. Details on Inequalities Addressing s

For all the intrusion tolerant architectures, when considering $s > 1$ sites (without special constraints, so that the best strategy is to distribute as evenly as possible the variants among the sites), the following inequality assures that the architectures continues to behave as expected if 1 sites is disconnected:

$$k \geq \lceil \frac{n}{s} \rceil.$$

Of course, the value of n depends on the architecture and is reported in Table 2. For instance, $n = 2f + k + 1$ for NVP and then (exploiting the definition of the ceil function)

$$k \geq \frac{2f + k + 1}{s}.$$

Writing on the left-hand side only k brings

$$k \geq \frac{2f + 1}{s - 1},$$

and imposing that k must be an integer results in writing

$$k \geq \lceil \frac{2f + 1}{s - 1} \rceil.$$

When the r variants under rejuvenation are also considered, the value of n reported in Table 4 are obtained.

Similar manipulations apply for the other architectures. Only SCP requires $s > 2$.

Appendix II. Self-Checking Programming with Voter

Consider a group of g variants in each self-checking component. A simple voter assesses whether there are $\lceil (g+1)/2 \rceil$ agrees among the results. Figure 60 illustrates the case $f = 1$ (and then $g = 3$) and $k = 2$ that requires $n = 6$ variants.

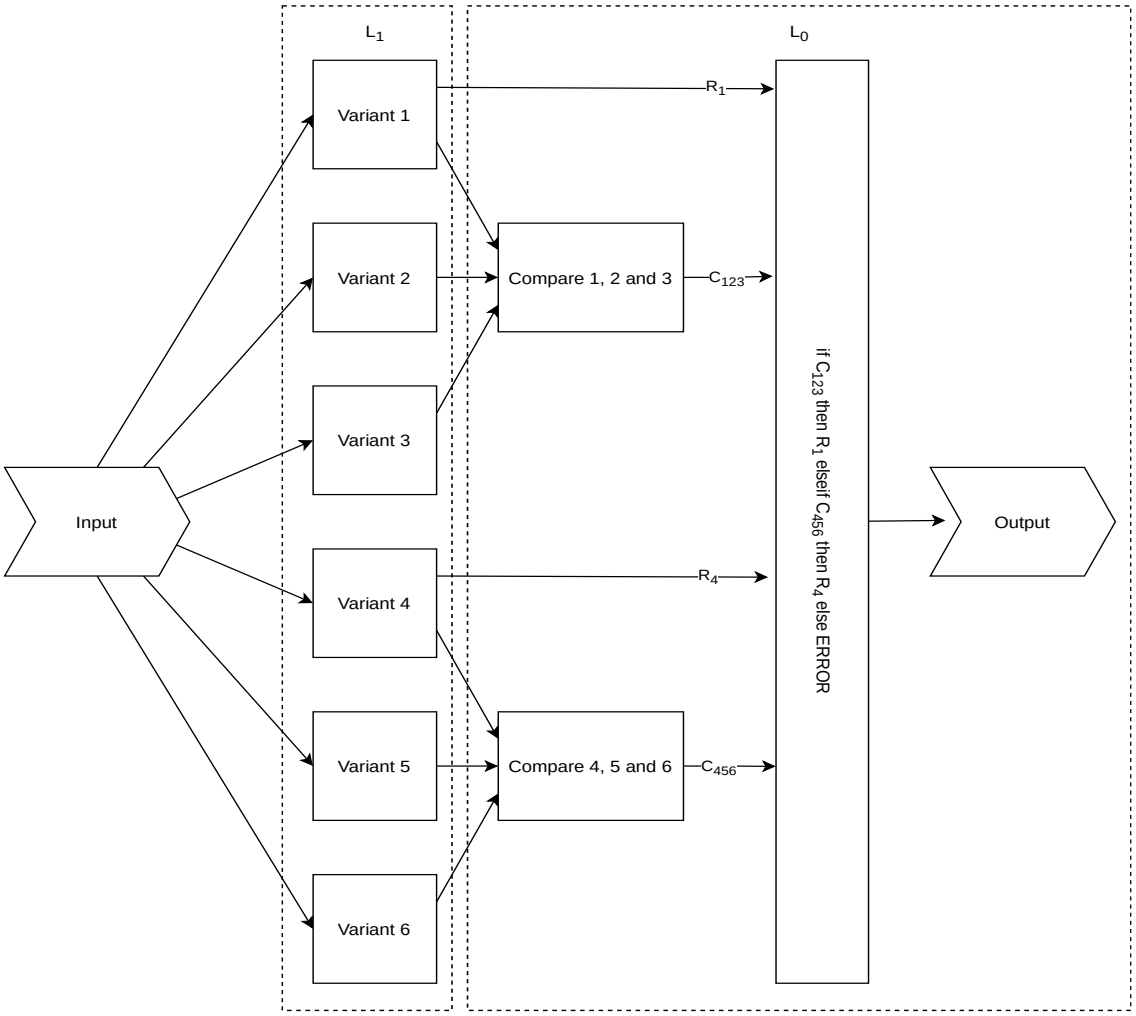


Figure 60 - SCPV with g=3 and n=6.

Where there are f value failures and no omission ($k = 0$), $g = 2f + 1$ guarantees a correct output because $\lceil (g+1)/2 \rceil = f + 1$, so $n = g \cdot n_{SC} = (2f + 1) \cdot n_{SC}$. Actually, in this case n_{SC} can be chosen equal to 1, and the architectures reduces to an NVP.

When there are f value failures and k omissions, the worst scenario is when there is a self-checking component with f value failure and 1 omission (so there is no majority), $n_{SC} - 2$ self-checking components with $f + 1$ omissions each (no majority) and 1 self-checking component with f omission (there is a majority of correct results), as represented in the following for $f = 2$ and $k = 12$, where i is an intentional failure, o an omission and c is a correct result.

SC_1	SC_2	SC_3	SC_4	SC_5
i	o	o	o	o
i	o	o	o	o
o	o	o	o	c
c	c	c	c	c
c	c	c	c	c

In this case, $k = (f + 1)(n_{SC} - 2) + f + 1$ and then

$$n_{SC} = \lceil \frac{f + k + 1}{f + 1} \rceil,$$

that corresponds to

$$n = g \cdot n_{SC} = (2f + 1) \cdot \lceil \frac{f + k + 1}{f + 1} \rceil + r.$$

The problem is that SCPV requires a huge number of variants compared with NVP. For instance, the previous case requires 25 variants whereas NVP with $f = 2$ and $k = 12$ requires 17 variants.

10. Reference

[1] Accenture Technology Vision, "The Post-Digital Era is Upon Us, are you ready for what is next?," Accenture, 2019.

[2] A. Avizienis, J. 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, 2004.

[3] X. Yu, C. Cecati, T. Dillon and M. Simoes, "The New Frontier of Smart Grids," *IEEE Industrial Electronics Magazine,* vol. 5, no. 3, pp. 49-63, 2011.

[4] M. Batty, K. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis and Y. Portugali, "Smart cities of the future," *The European Physical Journal Special Topics,* vol. 214, no. 1, pp. 481-518, 2012.

[5] X. Liu, C. Qian, W. Hatcher, H. Xu, W. Liao and W. Yu, "Secure Internet of Things (IoT)-Based Smart-World Critical Infrastructures: Survey, case study and research opportunities," *IEEE Access,* vol. 7, pp. 79523-79544, 2019.

[6] T. Kelly, Arguing Safety — A Systematic Approach to Managing Safety Case. PhD Thesis., York, UK: University of York, 1998.

[7] T. Kelly, "A Systematic Approach to Safety Case Management," *SAE Transactions,* vol. 113, pp. 257-266, 2004.

[8] M. Hsueh, T. Tsai and R. Iver, "Fault injection techniques and tools," *Computer,* vol. 30, no. 4, pp. 75-82, 1997.

[9] H. Ziade, R. Ayoubi and R. Velazco, "A survey on fault injection techniques," *International Arab Journal of Information Technology,* vol. 1, no. 2, pp. 171-186, 2004.

[10] K. Hayhurst, D. Veerhusen, J. Chilenski and L. Rierson, "A Practical Tutorial on Modified Condition/Decision Coverage," NASA Langley Technical Report Server, 2001.

[11] S. Kabir, "An overview of fault tree analysis and its application in model-based dependability analysis," *Expert Systems with Applications,* vol. 77, pp. 114-135, 2017.

[12] A. Segismundo and P. Augusto Cauchick Miguel, "Failure mode and effects analysis (FMEA) in the context of risk management in new product development: A case study in an automotive company," *International Journal of Quality & Reliability Management,* vol. 25, no. 9, pp. 899-912, 2008.

[13] B. Kordy, L. Pietre-Cambacedes and P. Schweitzer, "Dag-based attack and defense modeling: don't miss the forest for the attack trees," *Computer Science Review,* Vols. 13-14, pp. 1-38, 2014.

[14] C. Schmittner, T. Gruber, P. Puschner and E. Schoitsch, "Security Application of Failure Mode and Effect Analysis (FMEA)," *In International Conference on Computer Safety, Reliability, and Security,* pp. 310-325, 2014.

[15] E. Cioroaica, S. Kar and I. Sorokos, "Comparison of Safety and Security Analysis Techniques," *In Computational Intelligence in Security for Information Systems Conference,* pp. 234-242, 2021.

[16] D. Forster, C. Loderhose, T. Bruckschlogl and F. Wiemer, "Safety goals in vehicle security analysis," *17th ESCar Europe: Embedded Security in Cars,* pp. 74-88, 2019.

[17] A. Cook, H. Janicke, R. Smith and L. Maglaras, "The industrial control system cyber defence triage process," *Computers & Security,* vol. 70, pp. 467-481, 2017.

[18] R. Bloomfield and J. Rushby, "Assurance 2.0: A Manifesto," arXiv preprint, 2020.

[19] R. Wei, T. Kelly, X. Dai, S. Zhao and R. Hawkins, "Model-based system assurance using the structured assurance case metamodel," *Journal of Systems and Software,* vol. 154, pp. 211-233, 2019.

[20] R. Hawkins, T. Kelly, J. Knight and P. Graydon, "A new approach to creating clear safety arguments," *Advances in systems safety,* pp. 3-23, 2011.

[21] The Assurance Case Working Group, "Goal Structuring Notation Community Standard (Version 3)," Safety-Critical Systems Club C.I.C., York, UK, 2021.

[22] E. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering,* pp. 226-235, 1997.

[23] A. Cailliau and A. van Lamsweerde, "Assessing requirements-related risks thorugh probabilistic goals and obstacles," *Requirements Engineering,* vol. 18, no. 2, pp. 129-146, 2013.

[24] P. Fenelon, J. McDermid, M. Nicolson and D. Pumfrey, "Towards integrated safety analysis and design," *ACM SIGAPP Applied Computing Review,* vol. 2, no. 1, pp. 21-32, 1994.

[25] S. d. S. Amorim, F. Neto, J. McGregor, E. de Almeida and C. F. Chavez, "How has the health of software ecosystems been evaluated?: A systematic review," *Proceedings of the 31st Brazilian Symposium on Software Engineering,* pp. 14-23, 2017.

[26] E. Cioroaica, T. Kuhn and B. Buhnova, "(Do Not) trust in ecosystems," *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results,* pp. 9-12, 2019.

[27] E. Cioroaica, .. S. Chren, B. Buhnova, T. Kuhn and D. Dimitrov, "Reference architecture for trust-based digital ecosystems," *IEEE International Conference on Software Architecture Companion (ICSA-C),* pp. 266-273, 2020.

[28] S. Jones, "A discussion of issues and systems relevant to computer supported cooperative work," Stirling University, Department of Computing Science, Stirling, UK, 1990.

[29] I. Habli, W. Wu, K. Attwood and T. Kelly, "Extending argumentation to goal-oriented requirements engineering," *International Conference Conceptual Modelling,* pp. 306-316, 2007.

[30] S. Giordano, S. Vitiello and J. Vasiljevska, "Definition of an assessment framework for projects of common interest in the field of smart grids," JRC Science and policy report, 2014.

[31] S. vad de Hoef, Coordination of heavy-duty vehicle platooning, PhD Thesis, Stockholm, Sweden: KTH Royal Institute of Technology, 2018.

[32] R. Janssen, H. Zwijnenberg, I. Blankers and J. de Kruijff, "Truck platooning: Driving the future of transportation," TNO, Netherlands, 2015.

[33] D. Schneider and M. Trapp, "Conditional safety certification of open adaptive systems.," *ACM Transactions on Autonomous and Adaptive Systems (TAAS),* vol. 8, no. 2, pp. 1-20, 2013.

[34] J. Reich, D. Schneider, I. Sorokos, Y. Papadopoulos, T. Kelly, R. Wei, E. Armengaud and C. Kaypmaz, "Engineering of Runtime Safety Monitors for Cyber-Physical Systems with Digital Dependability Identities.," *In International Conference on Computer Safety, Reliability, and Security.,* pp. 3-17, 2020.

[35] A. Joshi, S. Miller, M. Whalen and M. Heimdahl, "A proposal for model-based safety analysis," *24th Digital Avoinics Systems Conference,* vol. 2, p. 13, 2005.

[36] E. Althammer, E. Schoitsch, H. Eriksson and J. Vinter, “The DECOS Concept of Generic Safety Cases - A Step towards Modular Certification,” *35th Euromicro Conference on Software Engineering and Advanced Applications,* pp. 537-545, 2009.

[37] S. Voss, B. Schatz, M. Khalil and C. Carlan, “Towards modular certification using integrated model-based safety cases,” in *Proceedings of the Internal Workshop on Verification and Assurance (Verisure)*, 2013.

[38] A. Retouniotis, Y. Papadopoulos, I. Sorokos, D. Parker, N. Matragkas and S. Sharvia, “Model-connected safety cases,” in *International Symposium on Model-Based Safety and Assessment*, 2017, pp. 50-63.

[39] D. Schneider, M. Trapp, Y. Papadopoulos, E. Armengaud, M. Zeller and K. Hofig, “WAP: Digital Dependability Identities,” *IEEE 26th International Symposium on Software Reliability Engineering (ISSRE),* pp. 324-329, 2015.

[40] G. Regan, F. Caffery, P. Paul, I. Sorokos, J. Reich, E. Armengaud and M. Zeller, “Securing a Dependability Improvement Mechanism for Cyber-Physical Systems,” *Advances in Software Engineering, Education and e-Learning,* pp. 511-522, 2021.

[41] J. Reich, I. Sorokos and M. Zeller, “An Eclipse Epsilon-Based Safety Engineering Tool framework for the Creation, Integration and Validation of Digital Dependability Identities,” *Model-Based Safety and Assessment,* p. 252, 2020.

[42] Alladi, Tejasvi, V. Chamola and S. Zeadally, “Industrial Control Systems: Cyberattack Trends and Countermeasures,” *Computer Communications,* no. 155, pp. 1-8, 2020.

[43] P. E. Verissimo, N. F. Neves and M. P. Correia, “Intrusion-Tolerant Architectures: Concepts and Desigm,” *Architecting Dependable Systems,* vol. 3, no. 36, 2003.

[44] K. Scarfone and P. Mell, “Intrusion Detection and Prevention Systems,” in *Handbook of Information and Communication Security*, 2010, pp. 92-177.

[45] A. Khraisat, L. Gondal, P. Vamplew and J. Kamruzzaman, “Survey of intrusion detection systems: techniques, datasets and challenges,” *Cybersecurity,* vol. 2, no. 1, 2019.

[46] P. Sousa, A. Bessani and R. Obelheiro, “The FOREVER service for fault/intrusion removal,” in *Proceedings of the 2nd workshop on Recent advances on intrusiton-tolerant systems*, 2008.

[47] T. Distler, “Byzantine Fault-tolerant State-machine Replication from a Systems Perspective,” *ACM Computing Surveys,* vol. 54, no. 1, 2022.

[48] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart and R. N. Wright, “From Keys to Databases Real-World Applications of Secure Multi-Party Computation,” *The Computer Journal,* vol. 61, no. 12, pp. 1749-1771, 2018.

[49] M. Rodriguez, K. A. Kwiat and C. A. Kamhoua, “Modeling fault tolerant architectures with design diversity for secure systems,” in *IEEE Military Communications Conference (MILCOM)*, 2015.

[50] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, 2005.

[51] A. Ceccarelli, A. Bondavalli, B. Froemel, O. Hoeftberger and H. Kopetz, “Basic Concepts on Systems of Systems,” in *Cyber-Physical Systems of Systems: Foundations -- A Conceptual Model and Some Derivations: The AMADEOS Legacy*, 2016, pp. 1-39.

[52] B. Littlewood and L. Strigini, “A discussion of practices for enhancing diversity in software designs,” Centre for Software Reliability, City University, DISPO-LS-DI-TR-04-V1-1d, 2000.

[53] A. S. Nascimento, C. M. F. Rubira, R. Burrows and F. Castor, "A Systematic Review of Design Diversity-Based Solutions for Fault-Tolerant SOAs," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, 2013.

[54] L. L. Pullum, Software Fault Tolerance Techniques and Implementation, 2001.

[55] B. Randell and J. Xu, "The evolution of the recovery block concept," in *Software fault tolerance*, 1995, pp. 1-22.

[56] F. Di Giandomenico and L. Strigini, "Adjudicators for diverse-redundant components," in *Proceedings Ninth Symposium on Reliable Distributed Systems*, 1990.

[57] A. Avizienis, "The N-Version Approach to Fault-Tolerant Software," *IEEE Transactions on Software Engineering,* vol. 11, no. 12, pp. 1491-1501, 1985.

[58] B. Randell, "System Structure for Software Fault Tolerance," *IEEE Transactions on Software Engineering,* Vols. SE-1, no. 2, p. 220–232, 1975.

[59] R. Obelheiro, A. Bessani, L. Lung and M. Correia, "How Practical are Intrusion-Tolerant Distributed Systems?," Department of Informatics, University of Lisbon, DI-FCUL TR 06–15, 2006.

[60] I. Gashi, A. Povyakalo and L. Strigini, "Diversity, Safety and Security in Embedded Systems: Modelling Adversary Effort and Supply Chain Risks," in *12th European Dependable Computing Conference (EDCC)*, 2016.

[61] M. Khan and A. Babay, "Toward Intrusion Tolerance as a Service: Confidentiality in Partially Cloud-Based BFT Systems," in *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN21)*, 2021.

[62] T. Dohi, K. S. Trivedi and A. Avritzer, Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions, 2020.

[63] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves and P. Verissimo, "Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery," *IEEE Transactions on Parallel and Distributed Systems,* vol. 21, no. 4, pp. 452-465, 2010.

[64] J. Qiu, Z. Tian, C. Du, Q. Zuo, S. Su and B. Fang, "A Survey on Access Control in the Age of Internet of Things," *IEEE Internet of Things Journal,* vol. 7, no. 6, pp. 4682-4696, 2020.

[65] B. Hardekopf, K. Kwiat and S. Upadhyaya, "Secure and fault-tolerant voting in distributed systems," in *IEEE Aerospace Conference Proceedings*, 2001.

[66] L. Wang, S. Ren, B. Korel, K. A. Kwiat and E. Salerno, "Improving System Reliability Against Rational Attacks Under Given Resources," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 44, no. 4, pp. 446-456, 2014.

[67] A. Babay, T. Tantillo, T. Aron, M. Platania and Y. Amir, "Network-Attack-Resilient Intrusion-Tolerant SCADA for the Power Grid," in *48th Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN)*, 2018.

[68] P. Jalote, Fault tolerance in distributed systems, 1994.

[69] K. Kwiat, A. Taylor, W. Zwicker, D. Hill, S. Wetzonis and S. Ren, "Analysis of binary voting algorithms for use in fault-tolerant and secure computing," in *5th International Conference on Computer Engineering Systems*, 2010.

[70] J.-C. Laprie, J. Arlat, C. Beounes and K. Kanoun, "Definition and analysis of hardware- and software-fault-tolerant architectures," *Computer,* vol. 23, no. 7, pp. 39-51, 1990.

[71] A. Bondavalli, F. Di Giandomenico and J. Xu, "A Cost-Effective and Flexible Scheme for Software Fault Tolerance," *Computer Systems: Science & Engineering,* vol. 8, 1993.

[72] M. R. Lyu, Software Fault Tolerance, 1995.

[73] R. K. Scott, J. V. Gault and D. F. McAllister, "The consensus recovery block," in *Total System Reliability Symposium*, 1985.

[74] F. Di Giandomenico and G. Masetti, "Basic Aspects in Redundancy-Based Intrusion Tolerance," in *14th International Conference on Computational Intelligence in Security for Information Systems and 12th International Conference on European Transnational Educational*, 2021.

[75] MISRA Consortium, "Guidelines for Automotive Safety Arguments," HORIBA MIRA Ltd., Nuneaton, UK, 2019.

[76] J. Reich and M. Trapp, "SINADRA: Towards a Framework for Assurable Situation-Aware Dynamic Risk Assessment of Autonomous Vehicles," *16th European Dependable Computing Conference (EDCC),* pp. 47-50, 2020.

[77] J. Reich, M. Wellstein, I. Sorokos, F. Oboril and K. Scholl, "Towards a Softwar Component to Perform Situation-Aware Dynamic Risk Assessment for Autonomous Vehicles," *Communication in Computer and Information Science Dependable Computing - EDCC 2021 Workshops,* vol. 1462, 2021.

[78] A. Schmidt, J. Reich and I. Sorokos, "Live In ConSerts: Model-Driven Runtime Safety Assurance on Microcontrollers, Edge, and Cloud," *17th European Dependable Computing Conference (EDCC),* pp. 61-66, 2021.

[79] Y. Papadopoulos, I. Sorokos, J. Reich and R. Wei, "DEIS-Project Dissemination," 10 October 2019. [Online]. Available: https://deis-project.eu/fileadmin/user_upload/DEIS_Engineering_tools_for_creation__integration_and_maintenance_of_Digital_Dependability_Identities_V2_Whitepaper.pdf. [Accessed 25 November 2021].

[80] D. Steinberg, F. Budinsky, E. Merks and M. Paternostro, EMF: Eclipse Modeling Framework, Pearson Education, 2008.