UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

School of Electronics and Computer Science

A rigorous tool-supported methodology for assuring the security and safety of cyber-physical systems

by

Giles Howard

ORCID ID: 0000-0002-6879-8544

Thesis for the degree of Doctor of Philosophy

28/12/2019

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
School of Electronics and Computer Science

Doctor of Philosophy

A RIGOROUS TOOL-SUPPORTED METHODOLOGY FOR ASSURING THE SECURITY AND SAFETY OF CYBER-PHYSICAL SYSTEMS

by Giles Howard

The increased usage of cyber-physical systems in a number of domains poses a unique challenge: how can one be assured of both the security and safety of these systems? While there are a large number of methodologies in the literature for performing security analysis or safety analysis, many of these are not specific to cyber-physical systems and the challenges these pose. Attempts at producing methodologies for security & safety co-analysis have equally met difficulties in terms of reconciling the different approaches and terminology often used by the separate domains.

One solution involves the development of a systems theory-based model for understanding how safety, security & other emergent behaviours of systems can be framed and understood. Such an understanding can then be used in a systematic methodology for performing co-analysis in a structured and robust way.

This thesis presents a methodology called *Security-Enhanced Systems-Theoretic Process Analysis* (SE-STPA), based on an underlying model known as *Systems-Theoretic Accident & Attack Model and Processes* (STAAMP), which combines safety and security analysis into one unified co-analysis method. It represents an evolution on existing work in safety by Leveson [90] and attempts to address several shortfalls of the existing approach in regards to security. SE-STPA is presented with two case studies that were utilised to evolve the methodology into a mature state. Finally, this thesis presents a discussion on future improvements that could be undertaken to develop the methodology further.

Contents

List of Figures

List of Tables

Declaration of Authorship

I, Giles Howard, declare that the thesis entitled *A rigorous tool-supported methodology for assuring the security and safety of cyber-physical systems* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as: [62] and [61]

Signed:.....

Date:.....

Acknowledgements

Thanks are owed to my supervisors, particularly Professor Michael Butler, whose experience, measured advice and assistance have guided me through every success and failure.

Thanks are owed also to my friends, for their tolerance for every missed social event and birthday celebration. Their further endurance of my consistent complaining about the volume of work I have put on myself is also greatly appreciated.

To my wife, Katherine

Chapter 1

Introduction

Cyber-physical systems - those systems with a real-world physical component and "whose operations are monitored, coordinated, controlled and integrated by a computing and communication core" [6] - are increasingly finding a place in infrastructure across the world, as well as in domains such as vehicle design and healthcare. While these cyber-physical systems present opportunities to monitor and manipulate the physical world through computation and networking [11], there exists within this growing field a need to ensure safety and security of such systems. This is due to the fact that cyber-physical systems represent a synergy of software and hardware wherein any failure modes within the system may have profound effects on anything from the success of a continued medical treatment to the survival of a plane full of passengers. It is therefore clear that this category of computing systems require an extensive regime of design and formal assurance support to ensure that these opportunities are fully embraced while still maintaining the safety and security of this class of systems.

This represents a challenge - particularly on the side of the security of these systems - where historically cyber-security researchers are said to have not considered how attacks affect the estimation and control algorithms and ultimately, how attacks affect the physical world [27]. It is therefore clear that there is a requirement for a robust methodology to ensure that the need for both safety and security within cyber-physical systems is addressed.

One possibility for addressing the need to ensure both security and safety of a system is through the use of formal method techniques. Cai et al. concisely describe the potential benefits of formal methods as follows:

> *Formal methods is an effective way to improve the quality of large-scale software and reduce the cost of software development... formal methods also enable accountable validation and verification of the resulting system, which*

> *reduces or even eliminates possible defects in the software and thus better maintains safety of the system [26].*

This view is reinforced in the recently-published guidelines for systems security engineering [116], which states that "Security, like safety and other system quality properties, is an emergent property of a system." Since this emergent behaviour cannot be predicted through analysis at any level simpler than the system as a whole [46], the sheer complexity of this analysis means that traditional testing methods alone are insufficient for full and proper verification. Formal methods are one solution to this challenge as they utilise mathematical rigour in modelling and verifying system properties [146] which can provide a higher degree of confidence that the system behaviours are well-defined and understood.

Additionally, as the complexity of systems increase, it becomes ever-more important to ensure that security and safety are embedded into all stages of the system life-cycle, as the cost of addressing issues and deficiencies within a system grows dramatically larger as the system development proceeds [54]. This is another area which formal methods can provide benefits, as they enable the system model to be created and expanded on, and so security and safety can be considered continuously throughout the system life-cycle. [146].

1.1 Research questions

The aim of this thesis is to develop a co-analysis methodology to allow the safety and security of cyber-physical systems to be analysed and for meaningful constraints on system behaviour - sufficient to successfully mitigate against the security/safety issues - to be generated for that system. This methodology also makes use of formal method techniques to validate that the generated constraints can demonstrate their effectiveness on a formal model of the system. The methodology should also be applicable in as many life-cycle stages as possible, to ensure that safety and security issues can be rectified earlier in the system life-cycle and therefore in the most cost-effective manner.

The formal research questions addressed by this thesis are therefore:

RQ1: Can a methodology be produced to analyse the requirements and design of cyber-physical systems with an aim to improve both their security and safety?

RQ2: Can the methodology additionally leverage formal method techniques in order to provide assurance that the mitigations generated as a result of analysis are demonstrably successful in reducing or eliminating security and safety hazards?

RQ3: Can the methodology demonstrate its utility through application to case studies of differing sizes and complexity, ideally utilising case studies involving cyber-physical systems in multiple domains?

1.2 Approach

The approach undertaken within this thesis in order to further the above research questions is to explore existing methodologies for performing security, safety or combined analysis (otherwise known as 'co-analysis') within the literature. From this point, a methodology is proposed which seeks to combine some existing approaches with new ideas in order to meet each of the three research questions; **RQ3**, in particular, requires that the methodology be applied and lessons to be learned from each new application of the methodology. The major contribution of this thesis is therefore a methodology that answers each research question and represents a foundation for future work in the area of co-analysis of cyber-physical systems.

Furthermore, due to the multiple dimensions of cyber-physical systems, there is a need to ensure that while the 'physical' portion of the system is not causing harm through its interactions with the real world and the environment, that the 'cyber' portion is also not being used as a launchpad for further harm to networks and other information assets. These two concepts are heavily intertwined and thus this thesis seeks to provide a methodology that can address both security and safety as first-class citizens of the analysis in order to meaningfully address both aspects.

1.2.1 Research method

The research method chosen for undertaking this thesis was the case study method, which is a qualitative approach where a "contemporary phenomenon" is studied "in a real-life context" [119]. The case studies were to be utilised in an exploratory fashion in order to assess whether the developed methodology was capable of suitable consideration of both safety & security jointly, and how successful it was at both. In order to ensure that the methodology did not become highly effective at analysing one system to the detriment of others, a multiple-case case study approach was utilised where each case study was an instance of a cyber-physical system but drawn from a separate domain [17]. This approach can be seen to have significant 'face validity' [120] due to the fact that it involved the application of a methodology for co-analysis of cyber-physical systems to a number of case studies of individual cyber-physical systems.

This approach was chosen in preference to a quantitative research method as it enabled rapid development of the methodology, and lessons learned to be released in short order. An aspect of future work (as detailed in Section 8.2) proposes the use of quantitative

research methods to provide further assessment, validation and improvement of the methodology.

1.3 Contribution

The major contributions of this work can be summarised as follows:

1. A methodology (SE-STPA) has been produced which is based on an existing, robust technique for assuring safety in critical domains. This methodology is able to identify both safety and security issues in systems under consideration and aid in the development of mitigations against them. It also leverages an appropriate formal method to enable the mitigations to be tested against a formal model of the system to ensure they are meaningful and correct in addressing the underlying issue.
2. A theoretical model underpinning SE-STPA known as Systems-Theoretic Accident & Attack Model and Processes (STAAMP) which provides the theoretical basis for understanding the interactions between safety and security in a cyber-physical system.
3. An analysis of the shortcomings of the original technique (STPA) and theoretical model underpinning it (STAMP) in the security domain.
4. Validation and development of SE-STPA through applying it to two real-world case studies involving cyber-physical systems of varying sizes and complexity.
5. An analysis of the outcome from both applications of the methodology, as well as lessons learned and proposed future improvements to the methodology.

1.4 Structure of thesis

This thesis describes the development and application of the methodology. It does so over the course of seven further chapters, which are described below:

Chapter 2 - Literature review: This chapter seeks to provide an overview of the various methodologies and techniques available in the academic literature for undertaking and mitigating against security and safety risks, as well as performing such analysis jointly. It further presents a brief outline on a range of formal method techniques within the literature as this is also an aspect of the work.

Chapter 3 - Development of SE-STPA: This chapter seeks to outline the STPA/STAMP technique that has been chosen for adaptation into the SE-STPA, in addition

to presenting existing work that has been undertaken in adapting the STPA/STAMP approach for other domains/contexts, and the specific modifications of the original approach undertaken to create SE-STPA.

Chapter 4 - Methodology: This chapter presents the methodology and provides a high-level overview of each step, as well as some details from the smart meter case study as a running example. It further describes the iterative approach in creating the methodology in its current state, as well as tools that are used to underpin and support the analysis process.

Chapter 5 - Smart meter case study: This chapter presents the smart meter case study and the results of applying the methodology to it.

Chapter 6 - UAV case study: This chapter presents the multi-UAV case study and the results of applying the methodology to it.

Chapter 7 - Discussion: This chapter considers the outcomes of both case studies, as well as potential refinements that could be made in future applications of the methodology.

Chapter 8 - Conclusion: This chapter concludes with a detailed assessment of the contribution against the research questions, as well as an outline of future work.

Chapter 2

Literature review

This chapter explores the literature around the topics of security analysis, safety analysis and combined (or *co-analysis*) methodologies that seek to perform both security and safety analysis in a combined manner. It then progresses into considering formal methods and their application to performing analysis. Finally, it ends with a section on the relevant industrial safety and security standards governing cyber-physical systems and their lifecycles.

The selection of security, safety and co-analysis methodologies is not intended to be exhaustive; instead, the selection of methodologies is based on the following criteria:

1. The applicability of the methodologies to cyber-physical systems. This was considered through examining if a candidate methodology had any association with case studies in cyber-physical contexts, such as having been applied on projects, products or concepts for smart cities, medical devices, autonomous vehicles or similar. For some techniques, the focus was clear due to the initial presentation of the methodologies being within a cyber-physical context. For a number of pre-existing techniques, their widespread usage and primacy within security or safety analysis meant they were included without explicit reference to a cyber-physical case study.

2. The pedigree of the methodology/techniques. This led to a degree of mixing of more modern and up-and-coming techniques (such as attack-fault trees) with established techniques (such as HAZOPs). Many of the newer techniques are themselves based around 'lessons learned' with the more-established techniques, and so are able to demonstrate improvement on some of the limitations of the original techniques.

3. The novelty of the technique/methodology in relation to other techniques. The focus of this chapter was to present various *approaches* which exist for performing security, safety or joint analysis, and so a number of iterations/variations on techniques were not included in order to maintain this focus.

2.1 Security & privacy analysis methodologies

2.1.1 CORAS

The CORAS risk management methodology attempts to identify security risks through a series of workshops and modelling steps, with the end goal state being each security issue identified having risk treatment/reduction strategies proposed to the client in order to manage the security issues identified [50, 23]. The seven steps can be summarised as follows:

Step 1 - Introductory meeting: This step involves an introductory meeting with the client in order to scope the target system for analysis, the boundaries that may exist in terms of interdependent systems and what the client's goals are in terms of what they wish to get out of the analysis process.

Step 2 - High-level analysis: This step involves an initial analysis of the target, in order to identify high-risk assets or those requiring more in-depth analysis, and this process aids the analysts in performing a full analysis on the system at a later date. This allows targets to be modelled in terms of both their static and dynamic behaviours.

Step 3 - Approval: The final of the preparatory steps; this step ensures that the client is presented with the analysts understanding of all assets within scope and their behaviours. This also involves the overt identification of consequence and likelihood of security risks, the competing importance of various assets to be noted and other required aspects of a risk evaluation criteria to be finalised and agreed with the client.

Step 4 - Risk identification: This step involves the identification of specific risks to assets by users and technical representatives of the client. This means understanding how issues like failure to follow training, compromise of user credentials, etc. can all contribute to result in assets being lost or undermined and allows threat diagrams to be drawn to understand how all these aspects inter-relate.

Step 5 - Risk estimation: This step assigns each threat scenario a likelihood which come together to create an unwanted incident score. Each unwanted incident that interacts with some asset must also be given a consequence scoring, to indicate the risk involved to assets of each unwanted incident.

Step 6 - Risk evaluation: This step maps risk values within a risk matrix and presents this to the client for approval and any additional comments/adjustments.

Step 7 - Risk treatment: This step attempts to mitigate identified risks with *treatments* which may be training, security-enhancing technologies, restrictions on access to aspects of the system, etc. Existing models/diagrams are to be updated with the treatments, and each treatment is to be given a cost/benefit assessment and presented to the client.

As can be seen, this is a very person-centric approach; many steps involve meetings and workshops, and the representatives of the client are involved in essentially every step. The approach attempts to make extensive use of those representatives of the client which use, maintain and have ultimate responsibility for the system. This permits analysts to work on a more complete view of the system and its potential security pitfalls than would perhaps be possible without this level of detail being actively exchanged and questioned.

CORAS utilises UML [118] for the purpose of modelling the system under consideration and borrows from many other methodologies, such as Hazard and Operability Study (HAZOP) [113] and Failure Modes and Effects Criticality Analysis (FMECA) [21], to inform its own perspective and approach [50].

While CORAS claims to be aimed at addressing *risk* in its many forms, case studies appear to primarily leverage it for security in IT-based systems [43, 137] thus it appearing in this section rather than in the later section on safety analysis.

2.1.2 LINDDUN

LINDDUN - an acronym based on the names of the underlying privacy threats it seeks to address - is a privacy-analysis methodology [40]. It seeks to address possible privacy issues within the context of a system in a highly systematic way, through explicitly mapping all data flows within a system using 'data flow diagrams' and attempting to mitigate identified privacy risks with known and tested *privacy enhancing solutions* .

Once again, this is a highly structured process, consisting of the following steps:

Step 1 - Define DFD: This step involves the identification of all data flows around the system within a data flow diagram (DFD); undertaking this step requires a solid understanding of the underlying system and therefore necessitates a complete high-level description of the system in order for the analysis to begin.

Step 2 - Map privacy threats to DFD: The next step involves overt mapping of *threat categories* to elements of the DFD, where appropriate. As privacy threats are explicitly defined by the methodology documentation, there are some inherent suggestions as to which types of elements of the DFD may experience which privacy threats, which can simplify the analysis.

Step 3 - Identify misuse case scenarios: This step considers the usage patterns and scenarios of the system to understand how the privacy threats may result in the degradation of privacy based on misuse of the system by insider and outsider actors. The generic threat tree patterns in the documentation for the methodology can therefore be applied and customised to suit the system in question and how the data and assets of the system are managed.

Step 4 - Risk-based prioritisation: The results of the preceding steps are prioritised to ensure that the highest privacy risks are addressed the most urgently, although the methodology makes use of no specific technique, leaving this up to the preference and experience of the analyst.

Step 5 - Elicit privacy requirements: This step considers the *positive* privacy goals that are implied by the privacy risks identified as part of the system's data flows.

Step 6 - Select privacy enhancing solutions: This final step aims to mitigate against the privacy risks by the use of various privacy-enhancing solutions which are suggested within the methodology's documentation and based on the experiences of the authors of the methodology.

As can be seen, this methodology takes a highly systematic approach to enhancing the privacy associated with data flows within a system, as well as aiming to mitigate through providing a number of suggestions at many of the steps of the methodology.

While privacy is not strictly related purely to security, maintaining confidentiality and integrity of information is an overlap of both security and privacy domains and this makes this methodology relevant in considering the existing work within the literature that is seeking to mitigate against malicious activity against systems. Much of the methodology is also high-level, concerning itself with data flows rather than specific aspects of architecture, and utilises generic elements within its documentation to enable the methodology to be applied as broadly across many domains as possible.

LINDDUN has found application in the RERUM smart cities project where it was used to elicit privacy risks [102] but also in more conventional architectural domains such as assessing the privacy risks inherent in an alumni relations system [5]. This would indicate a broad applicability to the domain under consideration, as this is also aimed at cyber-physical systems.

2.1.3 STRIDE

STRIDE, a threat modelling and analysis methodology, originates from Microsoft as part of their wider project on the Secure Development Lifecycle [55]. STRIDE seeks to mitigate against the following malicious behaviours:

1. ***S**poofing* - this is where an individual or system may intentionally misrepresent itself in order to deceive other further systems or individuals to interact with it beyond its actual status within a system. This end goal of this is often the acquisition of information that the entity, if behaving legitimately, would not have access to.

2. ***T**ampering* - Tampering of data seeks to undermine or manipulate data to change it from its correct, valid purpose towards some nefarious goals through modification to the data either at rest or while it is in transit.

3. ***R**epudiation* - This threat attempts to reduce confidence in the 'truthfulness' of data through tampering with logs or hiding the origin of an attack or piece of data. This allows attackers to operate while covering their tracks and reducing the chance of detection.

4. ***I**nformation disclosure* - This threat occurs when data that should remain available to a restricted number of parties or system is made available more widely, either intentionally or unintentionally. This may range from information that is helpful for debugging being available to curious parties, all the way up to databases of information being siphoned off by attackers.

5. ***D**enial of service* - This threat manifests when attackers are able to reduce the availability of a given service to normal, legitimate users through tying up network or processing resources. This results in legitimate users being unable to utilise the system for a period of time.

6. ***E**levation of privilege* - This threat is where an attacker is able to gain a higher level of access to the system than they should otherwise be allowed, either through exploiting vulnerabilities in access control systems or manipulating existing workflows to elevate one's credentials to a higher level.

STRIDE is similar to LINDDUN as the process involves the identification of data flows and data stores within the system using a data flow diagram (DFD) in order to enable easier identification of threats to the system. Each standardised *type* of entity when using the STRIDE methodology documentation also has a standard list of which of the above threats are most likely to apply to a given modelled entity - data stores tend to differ in threat model to a data flow, for example. The subsequent steps of the methodology then focus on mitigating the identified threats, and then validating that the mitigations are appropriate.

This approach is once again highly systematic through attempting to identify all threats across all aspects of the system, by completely modelling how data is transferred and stored within the system, and how the entities of the system interact with each other and with the aforementioned data.

STRIDE has been used across a number of domains: from applications of STRIDE to cyber-physical systems such as micro-grids and embedded healthcare systems [72, 127] to analysis of high-level networking topics such as OpenFlow [74] and software-defined networking components [10]. This would suggest it has applicability in the domains that this thesis attempts to address, and will therefore aid in informing the approach.

2.1.4 Summary

The three substantial methodologies considered in this section take highly systematic approaches to identifying and mitigating against security & privacy risks through a combination of guide-words, the explicit identification of the flow of information around the system, and an explicit identification of what actions attackers may take in reducing the security of the system. Each methodology also attempts to provide either a standard set of mitigations against these threats/attacks or a set of suggestions in focusing the analyst's efforts in both identifying and mitigating threats even where these may not be highly specific.

2.2 Safety analysis

This section will consider a selection of safety analysis methodologies which may either have a substantial pedigree in the area of safety analysis, or be particularly relevant to the cyber-physical domain.

2.2.1 HAZOP

Hazard and operability studies (HAZOP) are a structured approach to consider safety risks to a system, analysing all processes and sub-processes within a system for possible *deviations* with the help of a set of guidewords [36]. The use of the guidewords in the identification of deviations allows a consistent approach and each characteristic of each element of the system should have all possible guidewords applied to enable this to be as robust and complete as possible.

HAZOP also makes extensive use of documentation as analysis is undertaken; from the assumptions made in analysing the system, the scope of the analysis, all the way through to which exact characteristic and guide word have which consequences, and what safeguards can be applied in order to mitigate these consequences. Each safeguard is also assigned to an individual, to ensure that action is taken to actually ensure each safeguard is applied and integrated into the system operation.

HAZOP originated in the 1970s to handle the increased complexity of systems and the failure of existing techniques to handle it - the existing approaches primarily considered *equipment-oriented* practices rather than the increasing use of *process-oriented* practices [**Dunjo2010a**]. HAZOP has been extensively recommended through standards bodies, and many publications are associated with the technique over several decades [**Dunjo2010a**].

The limitations of HAZOP have been well understood for many years; one such paper aimed at addressing why HAZOPs sometimes do not reach their full potential dates from 1988 [99] and identifies 6 primary issues with how HAZOPs are often performed:

1. Lack of experience of the team - and especially of the study leader - in undertaking HAZOPs.
2. Failure to acquire sufficient detailed documentation around the system under consideration.
3. Inadequate management buy-in to the HAZOP process, or other managerial interferences that result in studies being rushed or otherwise incomplete.
4. Complacency around existing processes and procedures that are seen to not require review during the study, also known as 'blindspots'.
5. Shortage of technical information due to novelty or complexity of aspects of the system.
6. The 'ultimate limitation'; that the HAZOP team is a human and may fail to discover issues or hazards due to boredom or exhaustion.

McKelvey's identification of these six issues, as well as proposed solutions, indicate that even a technique with decades of use can be undermined relatively simply. This is due to the bulk of the effectiveness of the technique being drawn from human judgement, and so it is highly dependant on the HAZOP study team performing well with appropriate access to information. Attempts have also been made at automating HAZOP analyses and/or adding an expert system to reduce the human effort involved in performing a HAZOP analysis [144], which further suggests that the primary issue with the analysis technique is around its time-consuming nature and innate complexity. In-depth critiques of the technique, such as one by Baybutt [14], further emphasise the myriad issues that can degrade the effectiveness of the technique; these criticisms are related both to the people involved in a HAZOP as well as the underlying process.

2.2.2 FMEA/FMECA

Failure Mode and Effects Analysis (FMEA) is another technique for assessing and mitigating safety risks for systems of varying scales and, much like HAZOP, utilises a cross-disciplinary team of persons involved in the system to enable a robust approach to analysing a system [138].

FMEA uses this multi-disciplinary team to build an understanding of the system, as well as to collect data on known or suspected failures and their rate of occurrence. Ordinarily, the FMEA analysis will address sub-processes of the system in a prioritised

manner; either being provided by the system owner at the initial stages of the analysis, or identified through the analysis team. The team will then seek to consider each *failure mode* of the system in turn with an aim to reduce risk through meaningful safeguards.

FMEA can be considered to be more quantitative than other methodologies in this area of the literature as it focuses on three key metrics for each identified failure mode:

- Occurrence - the frequency of the failure mode.
- Severity - the risk to the overall operation of the system posed by the failure mode.
- Detection - the ability of the system to determine whether a failure mode has been entered into. This criterion is usually scored in the opposite manner to the other two, with a low score indicating a high certainty that the failure mode will be detected.

The multiplication of these three values together provides a Risk Priority Number (RPN) per failure mode, and enables the prioritisation of each failure mode against each other. Much like HAZOP, these RPNs should then lead to actions to be taken and responsibility/timeframes to be assigned to each action to ensure they are undertaken in a timely and accountable manner. Unlike HAZOP, these actions should then be continually assessed in such a way that the failure mode continues to have data collected in order to create a new RPN value. This new RPN value can then be compared with the original one to determine the success of the actions.

FMEA has been extended in several ways, with the development of an off-shoot known as Failure Mode, Effects, and Criticality Analysis being the most notable [22]. This particular adaptation of the underlying technique also concerns itself with *criticality* - the relationship that a failure may have when considered against the wider system purpose [39]. This tends to take the form of a matrix of probability mapped against criticality where criticality concerns itself with the impact on the system as a whole; ranging as an example from 'Undesirable' all the way up to 'Catastrophic'. A further extension of FMEA (into the security domain) was presented by Schmitter et al. [123] and is explored in Section 2.3.

Criticism has been levelled at the FMEA family of techniques, such as by Gilchrist who recommended improvements to how the methodology arrives at a Risk Priority Number (RPN) for faults as part of a deeper critique of the *ad hoc* nature of how RPNs are often generated during an FMEA analysis [53]. Further criticism of the effectiveness of RPN as a metric and how it cannot be used to meaningfully assess mitigations against identified failure is made as part of further suggestions of improvements to baseline FMEA by Puente et al. [108].

The use of FMEA/FMECA continues despite the first version of the technique being almost 70 years old and continues to be an integral part of several standards and prescribed processes in various industries [136], which speaks to the widespread applicability and utility of the technique.

2.2.3 FTA

Fault Tree Analysis (FTA) is another safety and reliability tool which enables the analysis of contributing causes and factors towards a single event of interest through a top-down approach [47]. Explicit identification of these contributing causes is enabled through the construction of a graph composed of basic boolean gates and events that all contribute towards the event under consideration [117]. The construction of fault trees and the subsequent analysis ordinarily consists of four steps [81]:

1. System definition.
2. Fault-tree construction.
3. Qualitative evaluation.
4. Quantitative evaluation.
5. Addressing underlying faults through adjustments to system design.

While the first step of system definition is common to most system analysis techniques, the process of constructing a fault tree is noted to often be highly manual and uses a common symbol definition across all domains [68]. The subsequent step of qualitative evaluation ordinarily results in minimal cut sets (MCS) [81] and a number of approaches for reducing a given fault tree to a minimal cut set have been proposed/developed [143, 76]. A minimal cut set enables the minimal set of contributing circumstances around a top-level event to be identified, which can focus the efforts and mitigations resulting from the fault tree analysis process.

The quantitative evaluation of fault trees is another area of active research. Quantitative evaluation of fault trees ordinarily results from the assignment of probabilistic scores to each contributing event/cause in order to determine the probability of the top-level event [145]. Many approaches have been presented to aid performing quantitative analysis on special cases or systems of high complexity such as work involving Markov Chains [20] or the use of priority AND gates [148]. Further approaches to enable modelling where exact probabilities are unavailable due to failure rates of components being unknown involve fuzzy fault trees [140].

Extensions to Fault Tree Analysis include Dynamic Fault Trees [29], which permits the modelling of timing across contributing events/factors, as well as state/event fault trees (SEFTs) [69], which seek to address state dependencies and temporal events.

2.2.4 Summary

As can be seen by the selection of three methodologies in this section, there are a variety of approaches available for performing safety analysis. Each of the selected techniques are well-tested, with all three having a history that stretches back at least 50 years, and involve substantial development and extension over that time-frame in order to account for the changes that have occurred within systems engineering within this time. There is also a contrast between HAZOP and its person-focused approach when contrast with FMEA and FTA, which are primarily numerical. There is also a mixture of approaches in terms of directionality: FMEA and HAZOP are both bottom-up approaches, while FTA is top-down in its analysis. This is because fault tree analysis considers a fault and then quantitatively analyses contributing causes, while HAZOP and FMEA are exploratory techniques which start by considering the low-level functionality and faults of aspects of the system and then attempt to determine if any of these component failures can contribute to a broader system failure.

2.3 Co-analysis methodologies

This section describes a selection of methodologies attempting to achieve a combined approach to performing both safety and security analysis. It does not seek to be exhaustive but to give a flavour of comparative methods within the domain of security and safety co-analysis.

2.3.1 CHASSIS

One example of a methodology that attempts to combine security and safety analysis into a unified framework is Combined Harm Assessment of Safety and Security for Information Systems (CHASSIS), developed by Katta et al. [71] and subsequently revised by Raspotnig et al. [112]. This framework aims to produce a unified process for safety and security assessments and analysis during the system development process.

It consists of three broad steps:

1. Elicitation of functional requirements.
2. Elicitation of safety/security requirements.
3. Specifying safety/security requirements.

The first phase of the assessment - elicitation of functional requirements - is carried out by the identification of users, system functions and services, through the drawing

of a diagrammatical use case (D-UC). Functions and services of the system are further understood by writing textual use cases (T-UC) and drawing UML sequence diagrams. The sequence diagrams (SD) defines components or 'objects' and their interactions, which are equally described in T-UCs.

The second phase - the elicitation of safety/security requirements - begins through the drawing of a misuse case diagram (D-MUC) using all of the items identified in the preceding phase. This step seeks to define misusers, harms and mitigations to the functions and services. The methodology then defines two distinct routes of analysis:

1. *For eliciting safety requirements:* the writing of textual misuse cases (T-MUC) and drawing failure sequence diagrams (FSD) which results in harm scenarios.
2. *For eliciting security requirements:* the writing of textual misuse cases (T-MUC) and drawing misuse sequence diagrams (MUSD) which also results in harm scenarios.

Once mitigations are identified, a trade-off analysis process is performed to assess mitigations against each other and acceptance criteria to ensure that mitigations are not contradictory between harm scenarios [111].

The third and final phase involves the translation of the harm scenarios into HAZOP tables from the T-MUCs, which enables the safety and security requirements to be specified and to provide a traceable artefact for other documentation to reference.

The CHASSIS methodology relies heavily on the existing analysis techniques by attempting to make them part of a unified methodology for performing both safety and security analysis; diagrams are based on UML [118] and the textual descriptions leverage the misuse cases technique [131]. Furthermore, the actual elicitation of misuse cases leverages HAZOP [113] as does the final step of actually specifying harm scenarios and their recommended mitigations.

CHASSIS can be considered favourably in many regards; its use of existing, well-explored techniques provides a degree of reassurance about the utility inherent in its approach. However, by opting to use existing techniques such as HAZOP, it also suffers similarly from the issues identified with the HAZOP approach within the literature [**Baybutt2015**].

CHASSIS has been applied in several case studies - one such example involves it being contrasted with the FMVEA approach in the context of automotive cyber-physical systems. In this case study, it was noted that the CHASSIS approach of separately considering safety and security had some weaknesses, such as CHASSIS being highly dependant on the knowledge of experts [122].

2.3.2 FMVEA

Another example of a co-analysis methodology is the Failure Mode, Vulnerabilities and Effects Analysis (FMVEA) methodology by Schmittner et al. [123] which takes the existing Failure Mode and Effective Analysis (FMEA) methodology for safety [9] and adds security/vulnerability analysis to it in order to produce a "unified model for safety and security cause-effect analysis".

FMVEA takes a bottom-up approach and attempts to integrate security into the standard FMEA-approach through the addition of a security-specific cause-effect terminology and analysis. Through making the security-specific language equivalent to the safety-specific language, a commonality is created between much of the standard FMEA life-cycle:

- **Failure causes** as safety terminology are viewed as equivalent to **vulnerabilities** for security terminology;
- Likewise, **failure modes** are deemed equivalent to **threat agents**.
- **Failure effects** are therefore equivalent to **threat modes**.
- **Failure severity** is therefore equivalent to **threat effect**.
- And finally, **failure criticality** has a security parallel in the form of **attack probabilities**.

This essentially creates a process similar to FMEA but with two internal streams, with one stream considering security vulnerabilities per component and the other considering failure modes. This can be seen in more detail in Figure 2.1.

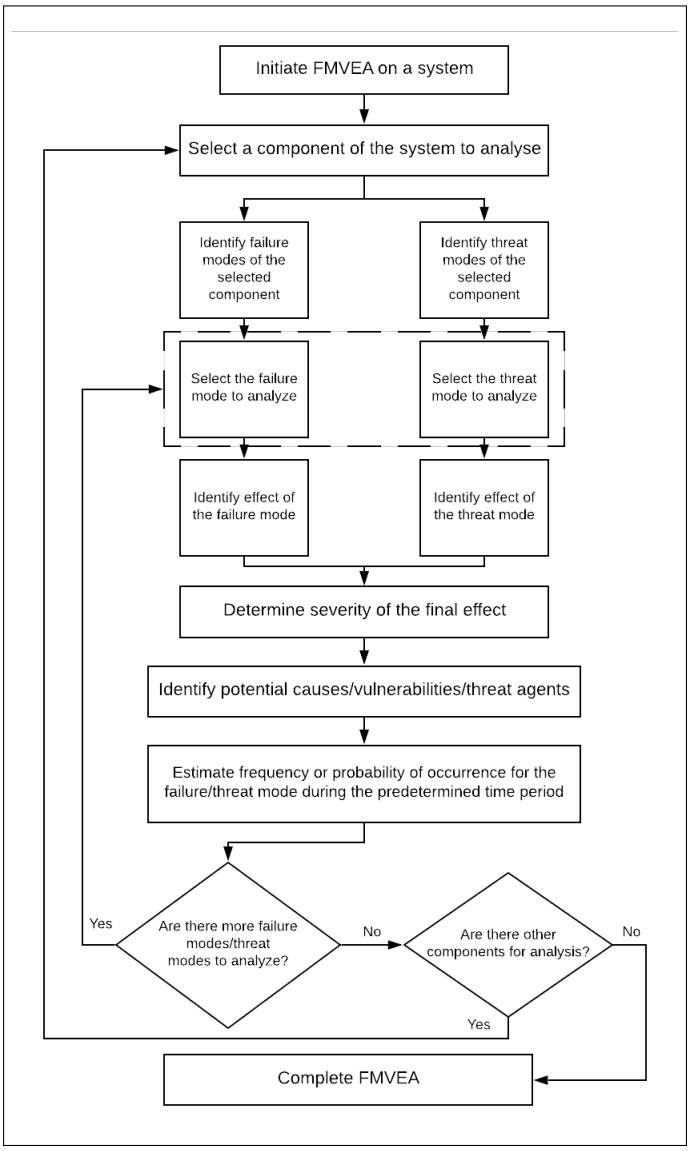

Figure 2.1: FMVEA analysis process [123].

FMVEA has been used in case studies by the authors to consider the safety/security impacts within an automotive cyber-physical systems context [121, 122]. It has also been applied as part of a combination of techniques for considering the security and safety issues which may apply to an autonomous boat platform [142]. One criticism drawn from this case study was that FMVEA was effective at determining hazards relating to single component failure or misuse, but was less capable at considering safety and security interactions that generally arise from autonomy or other complex systems. Furthermore, the approach was considered in some ways to generate fewer hazards than could be identified through comparative methodologies such as STPA or CHASSIS and could therefore be viewed as being less robust than these competing approaches. Credit was however given to the ease of use of the technique when considered to more in-depth techniques such as STPA/CHASSIS.

2.3.3 Attack-fault trees

Attack-fault trees represent an attempt to combine attack trees [126] from the security domain with fault trees [117]. This is undertaken in a formal manner [77] and attempts to augment the existing fault tree formalisms with some security-focused gate concepts. This approach differs substantively from FMVEA and CHASSIS as it attempts to be primarily quantitative; attack-fault trees permit the risk of both safety and security failures to be determined probabilistically through a stochastic analysis method, in order to provide a quantitative basis for decision-making by system owners.

This approach to formalising safety and secure risk permits several novel concepts to be explored. Kumar and Stoelinga [77] are able to model attackers (who have certain attributes such as 'risk appetite') to consider both the system as-is, but they can also perform a what-if analysis to consider the impact of attackers on the system with certain design changes implemented. Attackers and accidents can both be quantified in terms of cost and probabilistic risk and the interplay between malicious damage to the system, and whether this can undermine redundancy or other measures used to increase safety.

As this approach represents a fusion of both attack and fault trees, it comes from a pedigree in both domains; the fault tree technique has been used for many years and has a substantial body of research to support its use in safety analysis [117, 82], while attack trees are also well understood in the security literature, particularly with regards to cyber security [97]. This approach has not been utilised beyond any case studies provided by the author, however, and the attempt to quantify many aspects of security risk in terms of cost or disruption appears to rely on subject-matter expertise with no true basis of data to make certain determinations as part of the methodology.

2.3.4 Summary

This section does not seek to fully consider all co-analysis methodologies, but instead seeks to provide a general comparison of existing techniques. A broader and more systematic literature review can be found on the topic of safety and security co-analyses, such as that by Lisova et al. [94] or by Kriaa et al. [75]. What can be drawn from the co-analysis techniques/methodologies in this section is that all three of the co-analysis techniques appear to leverage an existing technique or approach, and then seeks to add consideration for another domain to it. FMVEA is a key example of this; the original technique (FMEA) is ordinarily used for safety and reliability analysis, but FMVEA has additional steps to enable it to be used to perform security analysis.

2.4 Methodologies combining formal methods and analysis techniques

This section explores the use of formal methods as applied to support safety, security or combined co-analysis techniques.

2.4.1 Formal methods in the safety domain

The use of formal methods in the safety domain is ordinarily concerned with specifying, modelling and verifying of correct and safe system behaviour for safety-critical systems, in order to support a safety case or other argumentation structure that the system is well-defined and only transitions between safe states in a predictable manner. The literature has a number of formal methods available to support these activities, a sample of which are considered in this subsection.

2.4.1.1 Z-notation, B-method and Event-B

The Z-notation permits the formal specification of systems to be undertaken. A formal specification describes 'what a system must do, without saying how it is to be done' [135] and therefore exists as a single source of truth for all parties involved in the system engineering process. It may also be viewed as a technical contract between client and engineers [146]. The Z-notation has been combined with automated theorem provers with an aim to producing reliable and unambiguous system specifications; one example of work in this area is given by Lockhart et al. in which system specifications in Z-notation were then passed to an automated theorem prover known as ProofPower in order to provide the consistency of the system specification under consideration.

A development from Z-notation, known as the B-method, is another approach to formal specification but using the mechanism of *refinement* to enable specifications to be made more concrete over time [4]. Refinement also generates a number of formal *proof obligations* which ensure that each refinement preserves constraints placed upon the previous level of abstraction. This means that an ever-increasing level of detail can be added to the specification as the system life-cycle proceeds, and enables a highly concrete model of system behaviour to be created which can then serve as the basis for implementation and testing, with a formal proof of correctness. One significant industrial case study of the B-method in use was to specify and formally verify the Météor project, a fully automated line on the Paris Metro system [15]. The result of this analysis demonstrated that the B-method could be utilised in the design and verification of a large-scale system with an aim to create zero-fault specifications, which is to say that no errors or faults were found in the resulting software during development or deployment, and therefore the specification can be deemed to be free of inherent faults.

The B-method has also been subject to its own extension known as Event-B, which utilises the B-method's approach to refinement but is focused around the notion of *events* which transition the system between *states* in order to enable richer, more detailed formal modelling of systems [3]. Case studies in various domains using Event-B [114, 25] have been undertaken, supported by the Rodin toolset for Event-B [58], primarily in support of providing formal demonstration of the safety of systems.

2.4.1.2 Symbolic Analysis Laboratory (SAL)

Another example of formal methods within the safety domain is given by Baronti et al. [12] in which formal verification techniques were applied to a novel charge equalization circuit design for lithium-ion batteries in order to demonstrate the safety of their approach. In this work, the authors leveraged the Symbolic Analysis Laboratory [18] environment to verify both safety and liveness properties associated with their proposed system through the creation of an initial formal specification and subsequent model checking and verification. The use of this formal method enabled claims to be made which were supported by proofs generated by the SAL environment.

2.4.1.3 Other approaches

Other formal methods have been used in support of analysing and verifying safety-critical systems; one such example is Uppaal [80], which is based on the theory of timed automata and has been applied to case studies in real-time automotive systems [73], as well as more broadly to verify translated Functional Block Diagrams for safety systems [132]. Enhancements to Uppaal have also been proposed, such as that by Larsen et. al to permit Uppaal to be applied to non-deterministic real time systems [79].

A related area of research is into the use of formal methods to produce 'oracles' which are derived from formal specifications of the system, in order to serve as monitors of safety-related systems [13]. An oracle is created from the formal specification of the system and therefore can monitor that the system only ever enters known safe states, while flagging any behaviours that diverge from this set of states. Alloy is one example of a formal method that has been applied to support the creation of an oracle; specifically, a test oracle was created for a modelled train station with knowledge of all positive traces (sequences of events, ordered by time, that are known to be safe) in order to verify that all test-cases execute and produce traces known to the oracle [139]. Alloy has additionally found use in a Problem-Orientated Software Engineering (POSE) case study which explored an aerospace case study relating to the safe release of stores from an aircraft at altitude [96].

2.4.2 Formal methods in the security domain

Primarily, the state of the art in terms of the usage of formal methods and their associated tools with regards to security is focused on analysis of protocols. An example of this approach is by Alexiou et al. [8] who performed a formal security analysis of the NFC (near-field communication protocol) through modeling the protocol and then performing probabilistic model checking to determine the probability of carrying out replay attacks depending on various contextual parameters (such as packet size, timeout settings, etc). Another example of usage of formal methods in security with an emphasis on formal protocol analysis is work by Künnemann and Steel in which they constructed a model of the Yubikey secure token protocols and performed a variety of analyses in order to determine the platform's susceptibility to replay attacks [78]. There has also been work in attempting to understand the emergent security properties that arise when multiple provably 'secure' subsystems are combined. One such model is proposed by Datta et al. and attempts to demonstrate a reasoning that - so long as all subsystems fulfill an invariant (which itself is associated with a security property of the system) - then the system should be immune to a class of attacks that would require a failure in the given security property [37]. The longest-standing formal approach to security analysis originates in the form of the Dolev-Yao adversary model, which permits encrypted communication protocols to be formally modelled and analysed in a symbolic manner [44].

Another example of using formal methods in security is work utilising the Communication Sequential Processes (CSP) formal specification language which enables the description of parallel communicating processes with a formal notation which is suited to communication/message passing that is inherent in communication protocols [124]. CSP was originally proposed in 1978 by Hoare as a process calculus but has been expanded on in recent years [59]. Case studies considering the Needham-Schroeder protocol [125] and demonstrating anonymity of voting systems [100] have utilised CSP as part of specifying system behaviours, particularly around information exchange by actors, and therefore this is a clear example of the use of CSP as a formal method in the security domain.

The Common Criteria has also found itself paired with formal methods previously; an example of this is given by Morimoto et al. which attempts to verify the security characteristics of a system through specifying the system specification in Z and then determining whether a set of instantiated security functional requirements, which are equally specified in Z, are met by the system specification [101]. Further work by Qamar et al. has also followed a similar methodology in an attempt to formalise and verify that systems meet security properties through implementation in Z, and animation in the Jaza tool [109]. It is clear therefore that there has been some substantial work undertaken broadly towards the generation of functional security requirements and then automated verification that a given system meets these requirements.

Event-B has also been leveraged as a formal method in performing security analysis such as work by Hoang et al. [56] in which Event-B was used to model a sample access control scenario representing the basic functionality of a bank and its security properties. A further example of Event-B being used for security analysis involves a high-level theoretical analysis by Mu [103] in which the issue of information flow security was considered and a framework was proposed for reasoning about secure information flows when refining from abstract to concrete models. A final example is with work by Gawanmeh et al. [52] in which Event-B was used to formally verify secrecy within group key distribution protocols.

2.4.3 Formal methods in a co-analysis context

Formal methods have also been utilised to support co-analysis methodologies, however, the literature has fewer examples than formal methods being applied to either safety or security individually.

One example that has been previously mentioned is the stochastic modelling used by the attack-fault trees methodology [77] which is able to translate the attack-fault tree structure into a stochastic model which is subject to simulation and model-checking through the Uppaal SMC toolbox [38]. This approach bases itself off of a technique known as Statistical Model Checking [128] which seeks to address the tendency of model state-space to grow exponentially when modelling complex systems through the use of statistics-based reasoning to reduce the search space.

Another example of formal methods used as part of co-analysis involves the use of Integrated Behaviour Trees [149] to develop a system specification with safety and security requirements in such a way as to ensure these requirements do not conflict. The resulting Design Behaviour Tree can then be translated into other representations to enable model checking and verification, such as Symbolic Analysis Laboratory format. This enables formal verification using existing well-supported formats and languages.

A further example is the use of a variety of semi-formal methods such as UML [118] as part of the CHASSIS analysis methodology [71, 112]. Other semi-formal method use includes work by Nicklas et al. which makes use of SysML models [60] to encourage system design that is both safe and secure, and to ensure both facets of the system design are adequately captured [104].

Finally, formal methods such as Event-B [31] have been paired with the Systems-Theoretic Analysis Process methodology [88], which is covered in significantly more detail in Chapter 4.

2.4.4 Summary

Formal methods have been utilised in many different ways to support both safety and security analysis according to the literature; they have also been used to support co-analysis methodologies and techniques. Formal methods in this context have either been used to create formal specifications of systems, which can then form the 'technical contract' in understanding a given system, or they have been utilised to undertake formal verification or model checking against a model of the system with an aim to demonstrate that the system meets required safety and/or security characteristics. This indicates that formal methods can be used to support a combined safety and security analysis through specification and/or verification.

2.5 Industrial safety/security standards

This section will identify a selection of relevant standards from the security and safety domains which are relevant to the cyber-physical system context of this thesis.

2.5.1 IEC 61508

The IEC 61508 standard represents best practice for ensuring functional safety throughout the life-cycle of a safety-related system where these safety-related systems utilise electrical, electronic or programmable electronic (E/E/PE) components [32]. It also serves as a broader high-level framework upon which many sector-specific standards are designed and constructed [16], such as IEC 61511 (covering the lifecycle of Safety Instrumented Systems within a Process Industry context) [33] and IEC 61513 (covering the lifecycle of Safety Instrumented Systems within the Nuclear Industry) [34].

In summary, the standard provides seven parts, which cover the following areas:

1. General requirements.
2. Requirements for electrical, electronic or programmable electronic safety-related systems.
3. Software requirements.
4. Definitions and abbreviations.
5. Examples of methods for the determination of safety integrity levels.
6. Guidelines on the application of IEC 61508-2 and IEC 61508-3.
7. Overview of techniques and measures.

The standard covers all life-cycle phases, from requirements elicitation through to operation and maintenance. It seeks to make functional safety a core consideration at all stages of the V-model of system development, and has a high focus on traceability as verification between each phase of system development as a mandatory activity. It additionally provides a number of techniques and measures with which to demonstrate that software is functionally safe. Finally, it permits the assignment of Safety Integrity Levels (SIL), which are quantitative measures of failure rate for the hardware of a system; a given SIL level (ranging from 1 to 4) will additionally define a number of Highly Recommended and Recommended techniques and measures which should be fulfilled for software in order to make claims that the software of a given system meets this level.

The standard therefore covers both quantitative assessment of systems as well as qualitative measures that can be taken in situations where quantitative analysis would be inadequate (such as with newly authored software). It therefore emphasises systematic process and best practices to ensure safety throughout the system life-cycle - this includes granularity such as ensuring that personnel working across all system are suitably qualified to be performing in their role.

2.5.2 IEC/ISA 62443

IEC/ISA 62443 represents the functional security equivalent of IEC 61508. It concerns itself with ensuring functional security across the life-cycle of industrial automation and control systems (IACS) [35] and consists of four major parts [106]:

1. Terminology, concepts and models common to all parts of the 62443 standard.
2. Design and operation of robust IACS cyber security management systems.
3. System-level requirements for effective IACS cyber security.
4. Specific, component-level requirements for effective IACS cyber security.

In the same theme as IEC 61508, IEC 62443 defines Security Levels (SLs) in three categories - Target Security Levels, Capability Security Levels and Achieved Security Levels. Furthermore, IEC 62443 goes a step further and attempts to provide a quantitative process for determining the maturity of cyber security-related processes and policies in the form of Maturity Levels - a Maturity Level of 1 may correspond to a process that is extremely fluid and inconsistent in application, while a Maturity Level of 4 indicates a well-developed, consistent process that is subject to continuous improvement.

IEC 62443 is a standard which is less mature than IEC 61508 but seeks to replicate many of the approaches of the IEC 61508 standard, such as its approach to quantification of safety and a significant body of requirements across the system life-cycle to ensure safety.

2.5.3 ISO 13849

ISO 13849 [66] is a standard similar in principle to IEC 61508 but is more focused on system design and architecture. Like IEC 61508, ISO 13849 focuses on functional safety; it even provides a metric for measuring the maximum probability that a safety function may fail per hour in terms of Performance Levels (PL) which range from A to E. Furthermore, it provides a second metric known as Category, which allows a broader classification of systems based on the safety-related techniques and measures used within the architecture, as well as capturing the diagnostic coverage of the system. Performance Levels and Categories are inter-related; it is not possible to meaningfully claim a system meets certain Performance Levels without equally meeting a given Category, which itself is dependant on the architecture of the system.

This dual-classification system differs from the IEC 61508 approach which utilises Safety Integrity Levels (SILs) as the sole metric that can be used to judge a system. While SILs and PLs are reached in broadly the same way (through consideration of statistical failure rates and factoring in other relevant measures of system reliability), the Category level in ISO 13849 has no equivalent in IEC 61508.

A further difference arises in the scope of the standards. IEC 61508 focuses on whole-system through-life safety management, including the operation and maintenance stage of the lifecycle, while ISO 13849 is more prescriptive and focuses more substantially on the design of the system and architectures that can be demonstrated to be provably safe.

2.5.4 Common Criteria

The Common Criteria for Information Technology Security Evaluation (also referred to as simply the 'Common Criteria') represents a common set of requirements for the security functionality of information system products and for assurance measures applied to these products during a security evaluation [28]. These requirements can be utilised to provide formal assurance of a system's security through the generation of a document known as the Security Target (ST) which attempts to summarise the security properties of a system under evaluation. The Security Target may also reference and indicate compliance with what are known as Protection Profiles (PPs) which are effectively generic or template forms of Security Target documents for entire classes of systems such as 'firewall' or 'identification smart card'. These PPs therefore define a baseline set of security characteristics that a system in that class should possess at a minimum. More detailed information on the standard is available from the full Common Criteria standard [28].

This standard is useful for managing the security life-cycle of elements in Information Technology but is not specific to Operational Technology such as most cyber-physical systems considered in this thesis. It is however an aspect of the security standards landscape, and so has been included for completeness.

2.5.5 Summary

This section of the literature review identifies relevant, high-level standards for use in managing safety and security throughout the system lifecycle within the cyber-physical domain. There are a number of child standards which have not been considered in this section and are applied in specific domains (such as the ISO 26262 for the automotive domain); this section does not seek to be exhaustive in that regard. Instead, it seeks to provide a flavour for standards that are often discussed in reference to cyber-physical systems more broadly.

2.6 Summary of literature review

This literature review captures a number of approaches for dealing with security, safety and combined analysis. It also considers a selection of formal methods that have been utilised to support security/safety or combined analysis in the past. Finally, the literature review considers a number of prominent safety and security standards governing the life-cycle of systems within a cyber-physical context.

It is clear from the literature that there have been a number of attempts to meaningfully combine security and safety analysis in order to permit systems at all phases of the life-cycle to be analysed and their safety/security properties to be assured. It also appears that few formal method techniques have been deployed to support a co-analysis context, instead these formal method techniques appear to have been used for one domain or the other, but rarely in a unified way. As cyber-physical systems become increasingly prominent, it is likely that security & safety analysis will be in increasing use, and one route for validating the security and safety aspects of a system design and implementation is formal methods.

This approach of co-analysis, supported by formal methods, may benefit cyber-physical systems as they become more regulated by various standards such as IEC 61508 and IEC 62443 where the underlying spirit of the standard is to take a holistic, through-life view of the entire system life-cycle. Having an approach which can provide this view of both the security and safety aspects of a system will be increasingly useful in a highly-standardised environment and context.

Chapter 3

Development of SE-STPA

3.1 Introduction

This chapter presents the origin of the methodology presented by this thesis. It begins with consideration of the factors that led to the creation of the methodology, in addition to why an approach built from an existing methodology and model was selected. It then discusses the existing technique in detail, as well as other work undertaken to apply these models and techniques to other domains. Finally, it details the additions made in SE-STPA that represent improvements on the original technique.

3.2 Justification of approach

The choice to take the existing *Systems-Theoretic Accident Model and Processes* (STAMP) theoretical model and *System-Theoretic Process Analysis* (STPA) methodology and modify them to enable them to be applied for both security and safety analysis was driven by the following major considerations:

1. Existing expertise within the research community at the home institution.
2. Adaptability of the chosen methodology.
3. On-going work around STAMP and STPA.

Each of these are detailed significantly in their own sub-sections.

3.2.1 Existing expertise within the research community at the home institution

The choice of STPA was at least partially driven by existing work undertaken by Colley and Butler at the University of Southampton [31] which explored utilising the Event-B formal method to support STPA analysis by utilising the notions of refinement and proof that are provided by this formal method. This represented a bedrock for further work which this thesis sought to advance.

Furthermore, the University of Southampton is host to two relevant research groups which provided the requisite expertise and scholarly community to support the creation of SE-STPA:

1. **The Cyber-Physical Systems Research Group:** This research group focuses primarily on the engineering of robust systems and has a number of research staff with backgrounds in safety-critical systems and formal methods [133].
2. **The Cyber Security Research Group:** This research group works in a number of fast-moving research areas such as blockchains and provenance, in addition to extensive work in the areas of security & privacy [134]. It is additionally one of the Academic Centres for Excellence in Cyber Security Research which represents a tripartite collaboration between industry, academia and government in the domain of cyber security. [41].

3.2.2 Adaptability of the chosen methodology

The choice of methodology was also driven by a desire to utilise an existing, well-established technique from one of the domains under consideration, and modify it to additionally address the other domain. This was due to the literature review indicating that a substantial number of the co-analysis methodologies considered had taken this approach, and it was felt to be the most pragmatic way forward with this thesis.

For this reason, Leveson's work in STPA and STAMP [90] was chosen for modification, as this framework had the following characteristics:

1. STAMP, as a theoretical model for understanding systems, essentially takes a 'first principles' approach and therefore views safety as an emergent system property. This was desirable as it enabled meaningful consideration of security, which is a similarly emergent property, to be integrated in a cohesive way without displacing or otherwise impinging on STAMP's understanding of safety.
2. STPA, as an analysis technique based atop the STAMP model, utilises this understanding of emergent behaviour to then carry out a systematic analysis to draw out

possible safety concerns and generate constraints in mitigation. This is a desirable characteristic as the challenge represented by cyber-physical systems necessitates a systematic approach to both safety and security. It was felt that STPA could therefore also be extended to enable systematic consideration of both safety and security.

3. Previous attempts have been made at including security into STPA as explored in Section 3.4. However, these contributions to the literature are lacking in ways that as explored in Section 3.5, and so it was felt that applying STPA as part of co-analysis of safety and security could be approached in a more robust manner with thorough consideration of security as well as safety.

4. STPA has been used in a number of safety-related case studies, such as in safety-critical applications in collaboration with the Federal Aviation Administration of the United States [49] and the Japanese Aerospace Exploration Agency [65], which indicated that the technique had a demonstrable history of being successfully applied to perform systematic safety analysis.

3.2.3 On-going work around STAMP and STPA

A significant driver towards the choice of existing methodology was that there is an active research community around the STAMP model and STPA technique. A number of conferences occur annually which are dedicated to consideration of STPA and methodologies derived from it; examples of this include the European STAMP Workshop and Conference [70] and the STAMP Workshop that is held annually at the Massachusetts Institute of Technology (MIT) [87].

There are also a number of derived techniques and approaches that are detailed more significantly in Section 3.4 which are representative of the continued interest in, and application of, STPA and its derived methodologies.

Furthermore, the STPA technique itself has been developed in several stages, beginning with Leveson's publication of *Engineering a Safer World* [88], followed by the STPA Primer [86] and culminating in newer publications such as the STPA Handbook [91]. This indicates that the technique is under a continuous development by the authors and this factored into the choice of it as the methodology of choice for modification.

3.3 STPA & STAMP

This section will provide a brief overview of both STAMP (the theoretic model underpinning Leveson's work) as well as STPA (the analysis technique for meaningfully analysing system behaviour for hazards).

3.3.1 STAMP

STAMP - Systems-Theoretic Accident Model and Processes - departs significantly from the historical, linear accident models of Heinrich and his contemporaries [85]. In this accident model, *accidents* are not the result of a linear chain of events as it was often considered in the early days of accident analysis, but instead are viewed as resulting from *inadequate enforcement* of constraints on system behaviour [92]. Safety is therefore an emergent property which is a result of adequate control of the system through sufficiently complete constraints on system behaviour. Failure to adequately constrain the system can result in the system entering an unsafe state and thus, a certainty of causing an accident when worst-case environmental conditions occur.

Leveson's development of the STAMP model was based primarily around the work of Rasmussen and his systems-theoretic work on 'natural migration of activities towards the boundary of acceptable performance' [88]. This concept inter-relates the economic, technical and social aspects of modern systems and additionally attempts to frame accidents or other undesirable situations as resulting from various pressuring influences or 'gradients' (i.e. work pattern optimisations, management pressure towards efficiency) which gradually push a system into the undesirable state. Rasmussen further felt that the best and most promising mitigation against this tendency appeared to be through the identification and highlighting of the boundaries of acceptable operation such that actors involved with the system could ensure their behaviours did not exceed these boundaries [110] which is the lens through which Leveson's methodologies attempt to understand and analyse systems.

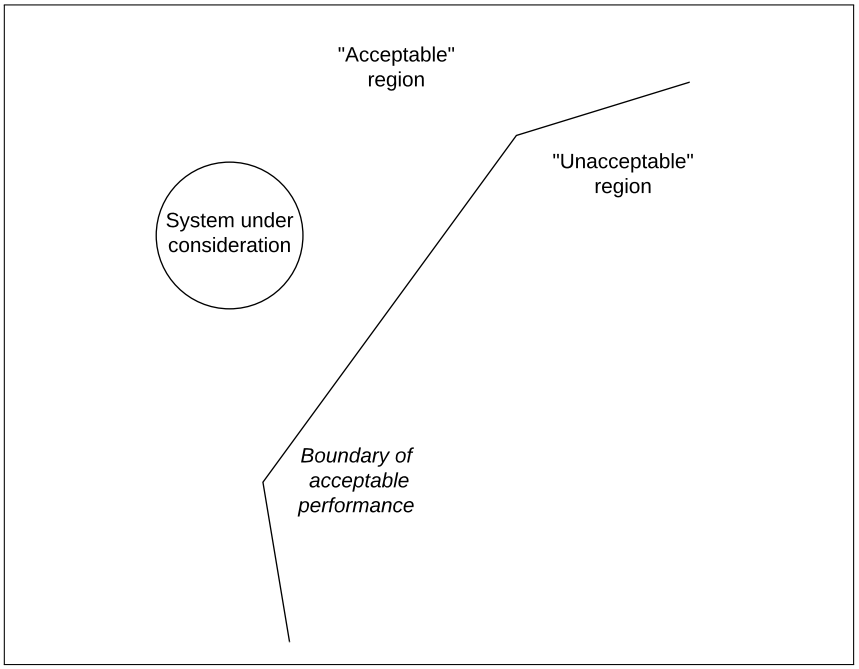

Figure 3.1: Visual representation of the "boundary of acceptable performance" concept, adapted from [110]

Leveson views that this notion of systems existing on a plane wherein actions can occur which bring them ever closer to the boundary of acceptable performance is a useful analogue to how system safety is maintained [89]. An ideal and safe system is designed such that a suitable degree of distance is maintained between itself and this boundary of acceptable performance; however, movement towards this boundary occurs when actors involved with the system make choices that trade safety for some other desirable characteristic (i.e. cost savings, reduction in maintenance burden, easier operability). The best method for ensuring a system remains safe is to therefore maintain adequate distance from this boundary by constraining system behaviour, as well as the behaviour of those who interact with the system, to ensure they cannot cause the system to migrate too close to this boundary and, ultimately, into the unacceptable performance region. In safety terms, this is what occurs during an *accident*. Furthermore, those interacting with a system should be made aware of why these constraints exist, and they should be carried through the entire system life-cycle [86]. A further benefit of this approach is that, as all systems have a degree of exposure to an environment that cannot be fully controlled, the maintenance of distance from this boundary means worst-case environmental conditions alone should not be able to push the system over the boundary.

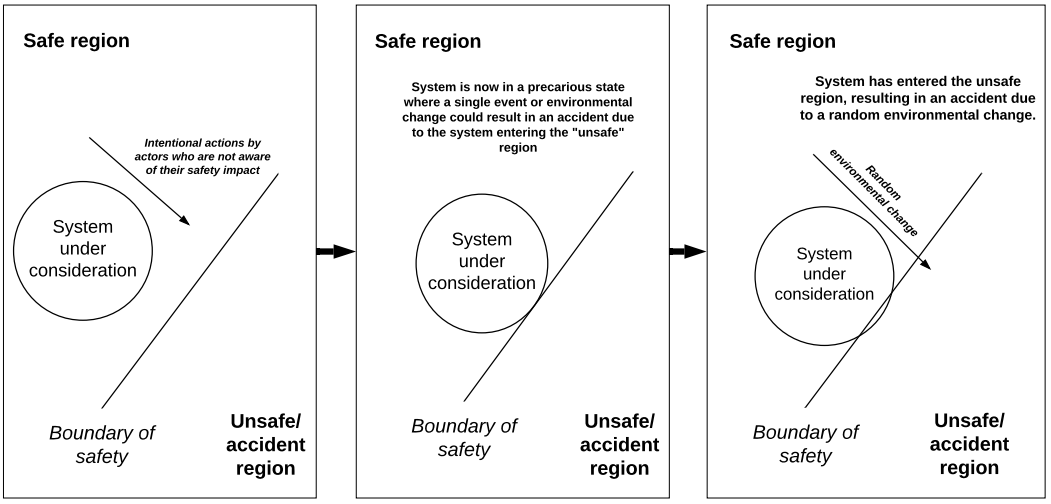

Figure 3.2: Visual representation of the 'boundary of acceptable performance' concept as applied to safety, adapted from [90].

Therefore, on a practical level, the STAMP model views that systems can be adequately modelled as interrelated components that are kept in equilibrium through feedback loops and control actions in order to maintain the system within the region of acceptable performance and therefore safe. Controllers are viewed as having a 'process model' which represents their understanding of the current state of the process they are controlling and they update this with information coming from a 'feedback' or sensor channel (which may correlate to multiple actual sensors in the system). They then 'act' on the process in response through some form of actuator. Any given system therefore consists of

a multitude of these controllers which are acting on each other and their individual processes in order to carry out the functions of the system. The emergent property of *safety* within STAMP is ensuring that these individual controllers maintain their process within a set of constraints, as well as ensuring that these constraints cover all potential states of the underlying process. There is an additional requirement that the process model for each controller is also detailed enough and that the sensor channels can read - to a sufficient granularity - the current process state and relay this information when necessary. Failure to meet any of the prior conditions can result in a natural progression of the system towards the boundary of unacceptable performance and therefore ever closer to an accident.

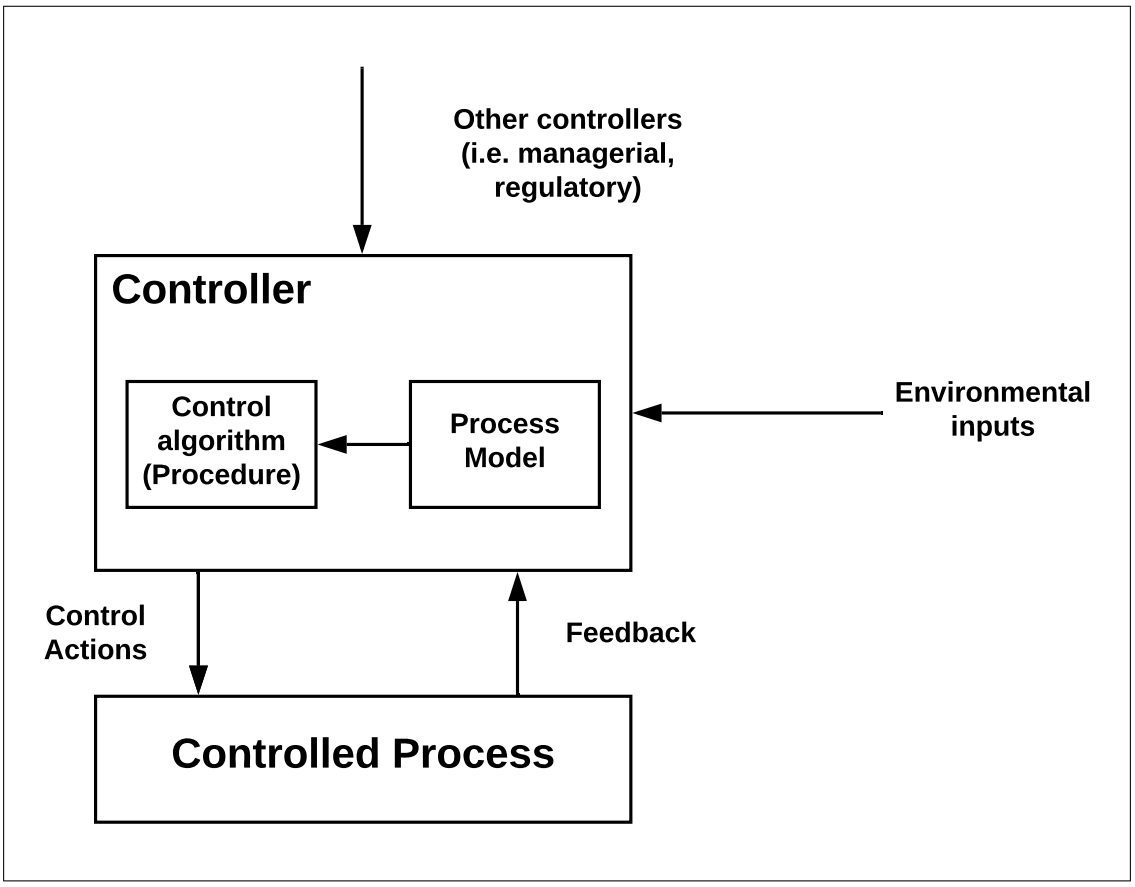

Figure 3.3: Visual representation of the controller paradigm defined by Leveson [86]

A 'safe' system therefore contains some number of controllers which all have a sufficient understanding of the underlying process that they are controlling and which can also react to maintain the state of their processes within some set of constraints. Safety is therefore a result of emergent behaviour and accidents are often the result of more than a single action or failure but instead require multiple failings across a series of control loops at a variety of levels across the entire system life-cycle; this is the fundamental difference between a systems theory-based approach as compared to the existing linear/causal

chain-of-events models embodied by the historical Domino Model and its derivatives [86, 88].

3.3.2 The STPA analysis process

Based on the STAMP understanding of systems, Systems-Theoretic Process Analysis (STPA) attempts to mitigate against accidents through identification and generation of constraints on control actions throughout a given system [88]. STPA consists of three major steps:

1. Begin by identifying the system engineering basis of the analysis:
   (a) Define the system purpose, which represents what the system exists to do.
   (b) Define the top-level accident and system-wide hazard states that will be considered over the course of the analysis.
   (c) Define safety constraints based off the hazards at this stage.
   (d) Draw a functional control structure for the system.
2. Identify potentially unsafe control actions and generate safety constraints in mitigation:
   (a) Control actions are analysed in each of the following four scenarios and if there is any risk of an hazardous state occurring as a result, that is recorded.
      i. A normally safe control action *is* provided and this creates an unsafe situation.
      ii. A control action required for safety is *not* provided.
      iii. A potentially safe control action is provided too early or too late; that is, at the wrong time or in the wrong sequence.
      iv. A control action required for safety is stopped too soon or applied too long.
   (b) For each of the identified unsafe control actions, generate a safety constraint which is simply a sentence representing some constraint on system behaviour such that the unsafe control action may not occur, to be integrated into either a future design or the implementation of the system.
3. Identify the causes of unsafe control; this involves understanding how combinations of control actions, or control actions being issued and not being followed, can lead to unsafe control. This is sometimes known as 'causal factors analysis' or 'scenario generation'. The result of this analysis may generate further safety constraints or necessitate the iteration of the system design if the unsafe behaviour is too complex to simply constrain within the current design.

The STPA technique can be applied at any point in the system life-cycle and therefore can be used either to design systems to be safe-by-design, or it can be used for analysis of existing systems with the aim of increasing their safety through the generation of constraints in order to highlight inviolable system limits. The STPA analysis supports this through being an iterable analysis process; the constraints generated in a previous analysis can be used to improve the design of the system before subjecting it to further analysis to validate that the new design is safer.

3.4 Developments of STPA/STAMP

One of the first substantial pieces of work to build on STPA was the STPA-Sec methodology proposed by Young [147]. This extension of the existing STPA methodology focuses tightly on the notion of 'mission assurance' and the security elements of a system under analysis. Through manipulating the terminology used in the analysis, it is possible to utilise STPA-based analysis to derive security vulnerabilities which may lead to loss states. The result of the analysis is a set of constraints which can then be used to iterate the system concept and design towards a more secure design.

Further methodologies - such as STPA-Priv - generalise the STPA approach once again, but this time towards privacy-based concerns. This is primarily realised through modification of the terminology once more to allow the privacy issues to be highlighted through the analysis; the notion of a 'loss' is instead substituted for 'adverse consequences' as this terminology is better understood in the field of privacy engineering [129]. An additional development is that, in place of mitigations, the focus on addressing privacy failings is instead addressed through implementation of 'privacy enhancing technologies' which serve to reduce the possibility of the identified adverse consequences occurring.

STPA-SafeSec is an additional methodology developed to address perceived shortcomings in the STPA-Sec methodology, which was criticised for viewing security as an issue only in relation to safety. STPA-SafeSec seeks to improve the capabilities of STPA-Sec by aligning terminology and viewing security and safety as equal concerns when performing the analysis [51]. Furthermore, STPA-SafeSec involves extensive analysis and a highly prescriptive approach to undertaking analysis on systems in order to ensure that all safety and security aspects are covered.

A final example is the work by Procter et al., which is named *SAFE and Secure* [107], that seeks to deeply integrate safety and security analysis and has roots in perceived shortcomings of existing methodologies such as STPA-Sec and STPA-SafeSec. It does so through a systematic identification of possible fault states a component or element of a system may have, and then creating an input-space compression in order to classify all possible inputs which may result in these faults. As this approach primarily focuses on the notion of inputs and their effect on the system, it can therefore be used to

consider both safety and security risks as fundamentally, it does not matter whether an input is accidentally incorrect, as is often the concern in safety, or whether it has been manipulated by an adversary, as is often the concern in security. The mitigations chosen to address these erroneous inputs are then chosen with a view to effectively constrain the inputs to only acceptable values; this may include the use of encryption, data integrity mechanisms, extensive checks, etc. A final analysis is then undertaken to analyse for undesirable internal behaviours which may equally result in fault states that can occur within the component itself. Any identified internal faults are then equally mitigated to ensure total coverage of the component, before another component is selected and the analysis started afresh.

It is therefore clear that the STPA approach has demonstrated itself to be quite malleable in the hands of researchers and has permitted refinement and development in a variety of ways as demonstrated by the methodologies detailed in this section.

3.5 Shortcomings of STPA/STAMP

The primary shortcomings of STPA and STAMP can be enumerated as follows:

1. A dependence on expert review during the analysis.
2. No built-in verification that generated constraints successfully mitigate against identified unsafe control actions, and therefore hazards.
3. Security considered only as a causal factor in relation to the safety of the system.

3.5.1 Expert review dependence

One of the major dependencies of STPA is a requirement for adequate subject-matter experts (SMEs) to be present during the analysis. This dependency is not isolated to STPA in particular; a HAZOP study is undertaken by a team which involves a number of subject-matter experts [63]. However, this usage of experts can be a detriment as much as a benefit. Even with the systematic approach provided by techniques such as HAZOP or STPA, much of the success of the activity is highly associated with the imagination, creativity and stamina of the team undertaking the analysis [14, 45]

There are essentially two types of experts normally present during hazard identification & analysis sessions when these take place in an industrial setting:

1. **System experts:** These experts have expertise in the system behaviours and so are useful when exploring how the system under consideration may become hazardous, especially as their expertise can be utilised to fill in gaps in existing documentation such as ambiguous requirements and/or design documentation. These experts tend to belong to the system creator.

2. **Domain experts:** These experts have broader expertise on the threat/hazard profile which may exist within the system's operating environment, as well as emerging threats/hazards.

The shortcoming of STPA is that it places a high degree of emphasis on the interaction between these two groups of experts, but in domains which are rapidly adopting cyber-physical systems, there may not be a significant number of *domain experts* who have a solid grasp of both the 'cyber' and 'physical' challenges within the domain and can articulate this in support of the analysis. This difficulty is magnified by the safety-focused nature of STPA, as it may therefore be difficult to 'tease out' security issues with the same degree of thoroughness.

In the baseline STPA approach, there is no clear guidance on how to address such a situation. Any methodology seeking to improve on this shortcoming of STPA would therefore need to provide some manner of understanding and analysing the system where these experts may not be available.

3.5.2 Checking of constraints

As STPA presents itself as an iterative analysis technique, the generated constraints as a result of identification of unsafe control actions (UCAs) do not undergo any *formal* validation. It is assumed that the expert input utilised during the analysis would inherently drive sensible constraints, and that the iterative intent of STPA would ensure that any issues with constraints would be addressed on a subsequent loop or during the re-design to integrate the constraints into the system design.

This represents a shortcoming in several dimensions:

1. The process of re-engineering the system is likely to be costly in terms of resource, and therefore it is essential to ensure that the design of the system is not built around constraints that are ineffectual at addressing the identified hazard.

2. It is entirely possible that generated constraints are contradictory or represent opposing behaviours when addressing different or complex hazards. This may not be detected until later in the system life-cycle, which would reduce confidence in both the technique and the remaining constraints. Furthermore, it may require a costly review of the design to ensure no other constraints are contradictory.

3. There is a practical 'tipping point' where the cost of undertaking further STPA analysis is seen to outweigh the benefit it may provide. This would mean that the constraints and any resulting design changes may be erroneous and there would not be a subsequent analysis undertaken to detect and address this.

Once again, in the baseline STPA approach, this dependency on continuous analysis does not mesh with the practical realities of engineering with finite time and resource. It is therefore clear that any methodology seeking to improve on STPA would be well served in addressing this shortcoming.

3.5.3 Consideration of security

The final major shortcoming of STPA relates to security; while the methodology does consider security as part of the analysis during the 'scenario generation' stage, this is purely considering security as it relates to safety, and not security in its own right. This may seem an irrelevant criticism when considering a *safety-focused* methodology, but for the intent of this thesis, this represents a substantial shortcoming.

Existing work that has already been explored, such as STPA-Sec and STPA-SafeSec, attempts to address this by either modifying the guidewords utilised to guide STPA analysis, or through integration of a number of well-regarded security techniques into the analysis. However, both of these approaches do not truly consider security *and* safety as part of a unified methodology, which represents a shortcoming when both of these aspects are of equal concern when considering the life-cycle of cyber-physical systems. STPA-SafeSec also involves a number of security analysis techniques which requires the system design to be finalised or for the system to actually have been implemented in some manner, before the security analysis can take place. This limits it to existent systems, and represents a late stage of the life-cycle to be undertaking security analysis.

This is the final major shortcoming of STPA; that security is essentially a 'second-class' citizen to safety within the framework of existing work within the STPA family of methodologies. A methodology seeking to improve on STPA in the context of cyber-physical systems should therefore consider security and safety in a meaningful, unified manner.

3.6 Iterative development of the methodology

Prior to the creation of the methodology and its associated model presented in this thesis, an initial phase of exploratory work was undertaken, followed by the usage of a candidate version of the methodology against a case study. The development of the methodology therefore can be seen to have occurred within 3 distinct phases:

1. ***The initial methodology:*** This stage involved a methodology that was minimally modified from standard STPA/STPA-Sec in its approach. A formal model of systems under consideration was built, but the relationship between generated *constraints* from the analysis and the formal model was unclear and poorly defined. This methodology also struggled to identify security issues that had no bearing on the system's safety.

2. ***The second version:*** This version of the methodology was clearer on the involvement of the formal model; dictating that a formal model was to be created prior to control action analysis being undertaken, and that the constraints generated during control action analysis were to be modelled in refinements of the formal model. This version attempted to integrate security analysis into the causal factors analysis step of STPA, which had limited success due to the fact that this made causal factors a very complicated step as the analyst had to both consider *residual safety issues* within the control loop through considering *how* control might be lost, as well as considering the bulk of all security issues such as manipulation of information in transit by malicious actors, etc. Causal factors analysis also occurred after the step integrating critical requirements into the formal model in this version of the methodology and therefore the bulk of security requirements could not be tested against the formal model without repeating the step a second time which was time-consuming. This version of the methodology was deployed against the smart meter case study, as detailed in Chapter 5.

3. ***SE-STPA:*** This is the final version of the methodology, as detailed in Chapter 4. The explicit identification of *adversaries* in its own distinct step ('adversary modelling') is designed to enable thoughtfulness around security risks, while the placement of this step prior to the integration of critical requirements into the formal model seeks to ensure that both the safety- and security-focused critical requirements can both be validated against the formal model to provide a higher level of assurance that they are reasonable and sufficiently-detailed in their approach. The unification of terminology such that all constraints - whether they come from causal factors analysis, adversary modelling or control action analysis - are known as *critical requirements* aids the consideration of both security and safety issues as first-class citizens of the analysis. This version was applied against the multi-UAV case study, as detailed in Chapter 6.

3.7 Creation of SE-STPA/STAAMP

The creation of Security-Enhanced Systems-Theoretic Process Analysis (SE-STPA) is intended to address the shortcomings detailed in Section 3.5, in addition to representing a robust methodology in line with the research questions as detailed in Section 1.1.

The methodology therefore integrates the following elements, above and beyond the baseline STPA approach:

1. The development of the existing STAMP theoretical model to include security to create *Systems-Theoretic Accident & Attack Model and Processes* (STAAMP).
2. The integration of a specific analysis step for security issues within SE-STPA, utilising unified terminology to ensure security is treated as a first-class citizen of the analysis.
3. The integration of the Event-B formal method into the analysis to provide a formal capability for validation and verification (V&V) of generated constraints.
4. Substantial modification of terminology utilised when compared to the baseline STPA approach.
5. Modification of several steps that are carried over from the baseline STPA approach.

Each of these developments from baseline STPA are explored and justified in their own section.

3.7.1 Expansion of the underlying systems-theoretic model

The first significant modification of the existing STAMP/STPA paradigm presented in this thesis involves the creation of the theoretical model known as *Systems-Theoretic Accident & Attack Model and Processes* (STAAMP). This new model integrates *attacks* into the broader framework of STAMP and therefore provides an understanding of security as an emergent property of systems. This section serves to present the development of this model from the baseline Systems Theoretic Accident Model and Processes (STAMP).

As previously described in subsection 3.3.1, STAMP approaches safety from Rasmussen's systems-theoretic perspective; namely that of the 'boundary of acceptable performance' which Leveson has applied to the domain of safety. Expanding this model to include security issues requires the addition of a further 'boundary of acceptable performance' which represents security as can be seen in Figure 3.4

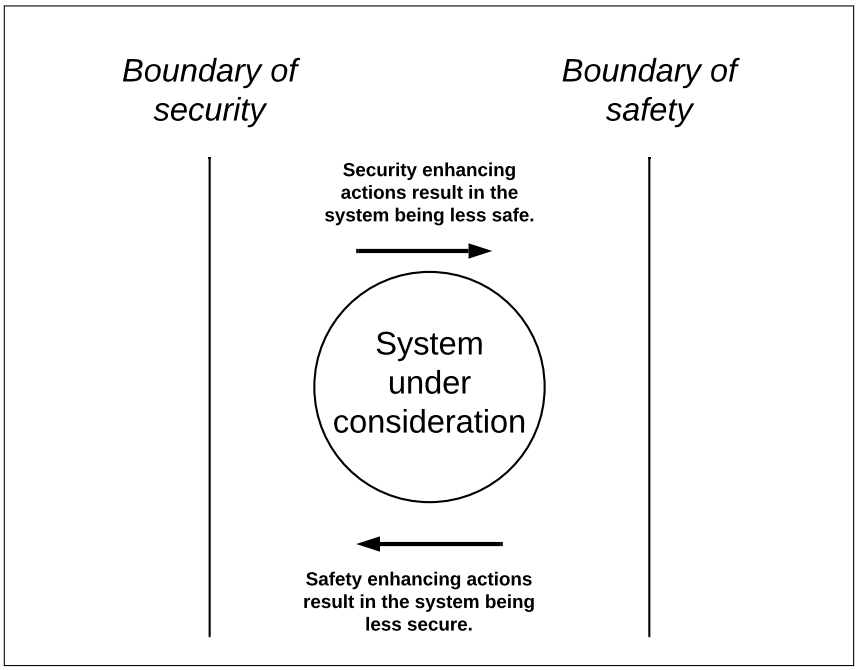

Figure 3.4: Simplistic representation of the 'boundary of acceptable performance' concept as applied to safety and security jointly

The fundamental issue with a model such as that represented by Figure 3.4 is that it fails to account for the fact that there are modifications to the design which can improve or degrade both safety and security in lockstep; i.e. that safety and security are not always trade-offs.

An example can be found in industrial control systems where role-based access controls are utilised to ensure that individuals may only control equipment that they are trained or certified to operate. This is beneficial from both a safety and a security perspective:

- From a security perspective, attempting to control aspects of the equipment will require authentication as a valid user with appropriate privileges. This prevents casual attackers and represents an additional layer of defence on top of any physical access controls that exist on the site. Furthermore, having user-level access controls ensures a degree of traceability and transparency in the case of a user misusing the equipment with an intent to disrupt or degrade the underlying process.
- From a safety perspective, there are likely to be safety consequences if someone with an inadequate understanding of the machinery and its limitations is permitted to operate the machinery. The management of user privileges prevents users from controlling aspects of the machinery they may not understand, which can reduce the risk of fire, energetic releases, etc.

While it may be possible to make an argument that this restricts wider operation of the equipment, which in some hazardous situations may be desirable from a safety

perspective, a majority of industrial control systems are also equipped with emergency stop functionality in the form of physical buttons placed near all machinery. This enables any user, regardless of role, privilege or training, to halt the machinery if they perceive a potentially dangerous situation emerging, while restricting control of the equipment in more general sense to those who are adequately qualified and experienced with the equipment.

A more mature and realistic model is therefore one that takes into account that while there are design choices influenced by safety which can degrade a system's security and vice-versa, there are also a number of design choices which can improve both security and safety. This is represented in Figure 3.5.

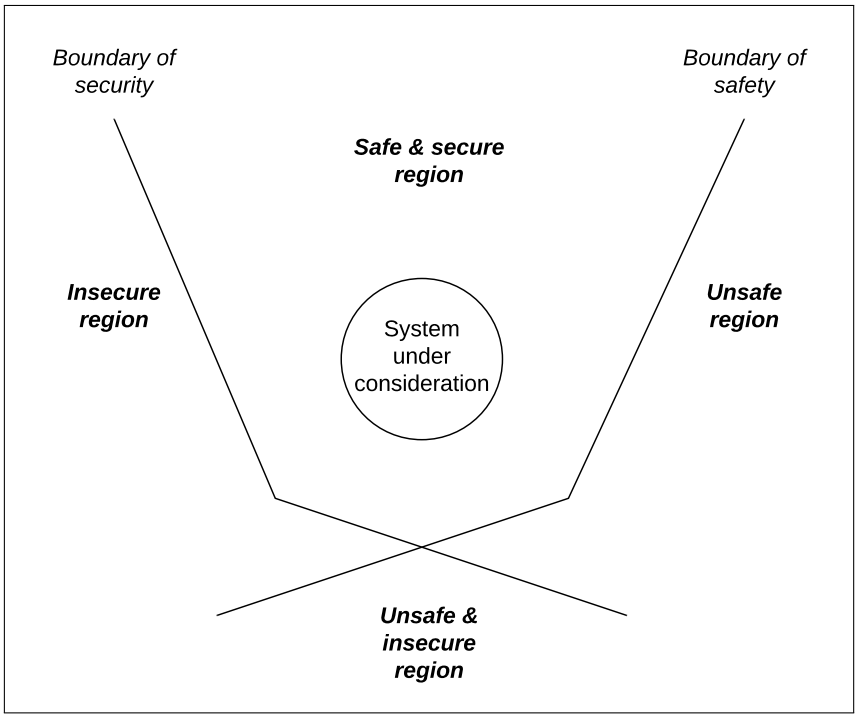

Figure 3.5: Advanced representation of the 'boundary of acceptable performance' concept as applied to safety and security jointly

The model as seen in Figure 3.5 represents several concepts:

1. There remain design choices or operational decisions which can result in the system drifting towards one undesirable region (i.e. the unsafe region) without getting closer to the other undesirable region (i.e. the insecure region).
2. On the other hand, there may be design choices or operational decisions that migrate the system away from both boundaries and therefore improve both the safety and the security of the system under consideration.
3. There exists an undesirable region where the system can be both unsafe and insecure. This might manifest through an attacker breaching the security of a system

(i.e. bringing it into the insecure region) and then subsequently utilising the system to cause direct harm through bypassing safety constraints (i.e. bringing it into the unsafe & insecure region).

This elaboration of the STAMP model includes security *on a theoretical level* and considers both *accidents*, which ordinarily result from a series of unintentional actions by benevolent actors, as well as *attacks*, which result from a combination of intentional actions by malicious actors in addition to unintentional actions by benevolent actors. The revised model presented in Figure 3.5 assumes that there is a strong likelihood that well-intentioned actors may occasionally make decisions that undermine the security of a system broadly, such as through reusing passwords or disabling security features that interfere with their interaction with a system. These unintentional actions can then form part of a general drift towards the insecure region, which a malicious actor can then exploit by taking further actions that push the system over into the insecure region. The end result of this is what would commonly be known as an 'attack', in the same way that a system entering the unsafe region is ultimately known as an 'accident'.

This view of the drivers behind accidents and attacks differs from the understanding presented by Young et al. [147] in which it was suggested that unintentional actions by benevolent actors were the leading cause of accidents, while intentional actions by malevolent actors were the primary cause of security issues. However, Young's understanding of the security and safety dichotomy is derived from a military domain and context, in which the discipline of individual users is higher than can reasonably be assumed when considering cyber-physical systems.

These revisions of the STAMP model to include security also results in a similar conclusion to that of Leveson [90], which is that the best defence against both accidents and attacks is the creation, implementation and maintenance of constraints that are known to users of the system throughout the system life-cycle. Through constraining users, maintainers and management, as well as the system behaviours, it is possible to maintain a significant 'distance' from the boundaries of security and safety, and to ensure that all who interact with the system maintain that distance.

This also applies to malevolent actors, who often rely on vulnerabilities or weaknesses in the system architecture to establish a foothold from which to take further actions to undermine the system and eventually successfully attack it. Through identification and analysis of any innate weaknesses of the system, as well as likely attack paths, these can be constrained and controlled to reduce the attack surface of the system to a manageable degree.

The revised model as presented in this section is therefore entitled **Systems-Theoretic Accident & Attack Model and Processes** or **STAAMP** to account for its consideration of security and safety.

3.7.2 Embedding security into STPA

As STPA uses STAMP as the lens through which systems are understood and considered, SE-STPA presented by this thesis equally utilises the Systems-Theoretic Accident & Attack Model and Processes (STAAMP) for its view on systems, as detailed in subsection 3.7.1. This adds the requirement for the analysis to consider security as a first-class citizen, as the underlying model views both security and safety as equal concerns. This is addressed through the addition of a security-focused step into the usual steps of applying STPA, which is known as *adversary modelling*.

Adversary modelling takes place subsequent to Control Action Analysis (Step 3 in 'standard' STPA) in SE-STPA as this ensures that the system model has been constructed in terms of controllers and controlled processes and that information exchanges between controllers have been identified. It seeks to provide an explicit capture that various categories of adversary and their possible attack paths through the system have been considered. As the major goal of analysing cyber-physical systems is ensuring that correct and safe operation occurs despite the additional attack surface represented by the cyber/networking components, the placement of this step subsequent to Control Action Analysis also ensures that the safety-relevant behaviours of the system have been thoroughly analysed prior to considering how adversaries may interfere with the system; this is important as it contextualises the correct, safety-related behaviour of the system prior to considering the impact that an adversary may have, and thus enables consideration of security issues which interfere with achieving these safety-related behaviours in addition to considering around security issues that do not interfere with the safety of the system.

Adversary modelling achieves this through identification of the adversary, their possible resources, and their intent with regards to the system (e.g., disruption, mere curiosity, etc). Attack paths are considered through labelling controllers and information links between controllers with 'manipulation points' and then allocating a number of manipulation points commensurate with the perceived threat of the adversary and their capabilities. Finally, constraints can be generated to either represent constraints on system and user behaviours or to inform design revisions to reduce the anticipated attack surface available to a given adversary. More detail on this step in practice can be found in subsubsection 4.2.4.6, but the overall intent is to provide an explicit capture of perceived malicious actors and their capabilities and constrain the system accordingly.

This approach was chosen over alternatives as many alternative approaches, such as undertaking hands-on penetration tests against perceived weaknesses of the system, would limit which life-cycle stages that SE-STPA could actually be utilised on a system. The STPA family of methodologies are intended for use throughout a system life-cycle with an aim to identify constraints as early as possible, and therefore it would be contrary to the expectations of a systems-theoretic approach if SE-STPA was only applicable once the system had been implemented. This is due to the STPA family of methodologies

seeking to *meaningfully* constrain system behaviour as part of design, and so the later the analysis is carried out, the less malleable the system design is likely to be, and the greater the risk of constraints not being integrated into the design in a meaningful way. This means that, in particular reference to adversary modelling, it trades some specificity in terms of the exact approach that may be taken by an adversary in order to have broader applicability throughout the system life-cycle.

3.7.2.1 Relationship of adversarial modelling to the literature

Adversary modelling as a concept is informed and inspired by a number of approaches from the literature, which are detailed below:

1. **Misuse cases** as presented by Sindre and Opdahl represent an explicit capture of a sequence of actions which can undermine the security of a system, as well as identification of other metadata involved in the misuse (i.e. pre-conditions, existing mitigations, etc) [131]. These serve as a counter-point to the *use cases* that are often used in software engineering requirements elicitation.

2. **Abuse cases** as presented by McDermott and Fox which take a threat actor-centric approach, considering the resources, skills and objectives of each actor involved in abuse/misuse of the system [98]. These actors are then considered in the context of an abuse case, where characteristics such as harm, privileges utilised and the active steps involved in the actor abusing the system are captured and considered.

3. **The Dolev-Yao adversary** represents a formal adversary with a significant degree of influence over communication protocols and channels [44]. While a *formal* adversary model is not used by adversarial modelling, it was an aspect of the literature that was considered during the development of the adversarial modelling step in SE-STPA .

4. **Attack trees** [126] were an additional source of inspiration for adversarial modelling. Once again, although adversarial modelling does not prescribe the use of attack trees, it does explicitly outline attack paths through the system, and so attack trees are one technique that could be utilised within the adversarial modelling step for providing a visual representation of an adversary's actions.

Adversarial modelling can therefore be said to primarily have been influenced by the notion of abuse/misuse cases. These approaches seek to inform the security design through explicit capture of aspects of a threat and/or an actor, and then provide an *argument* that the current security mitigations in the design are sufficient or, where inadequate, that additional requirements are generated to address this shortfall. Adversarial modelling is also intended to serve as the basis for an explicit *security* assurance case [7]; to

serve as evidence that all valid threat actors and their possible impact on the system have been analysed and addressed.

3.7.3 Integration of a formal model

The usage of the Event-B formal method [3], and the associated Rodin toolset [2], aids the methodology by providing a higher degree of assurance that identified constraints actually mitigate against potentially insecure/unsafe system behaviours. This is in direct response to the shortcoming of the baseline STPA approach which has no formal validation step to ensure generated constraints meaningfully address the associated undesirable behaviour, as detailed in subsection 3.5.2. Instead, STPA relies on the expert review involved in the analysis process to serve as the sole 'check' on generated constraints, which can lead to contradictory or conflicting constraints.

3.7.3.1 Event-B summary

This section of the thesis aims to provide the reader with a high-level overview of Event-B, sufficient to understand its role in the methodology. Terminology is sourced from Abrial [3] and Robinson [115] and can be found in Table 3.1.

Table 3.1: Event-B terminology

Term	Definition
Machine	Machines model the dynamic behaviour of the system. They can *see* a *context*, and contain a *state* as well as a number of *events*. The *state* of the system is represented through *variables*, which themselves are defined by either *invariants* or *theorems* within the model. A Machine may refine another Machine.
Context	Contexts are used to model constant values such as carrier sets, relations and functions as well as properties of those constants which are called *axioms*. Furthermore, *theorems*, which express properties of the constants that can be deduced from the axioms, are modelled within the context. A Context may extend another Context, through providing additional sets or constants.

Table 3.1: Event-B terminology (continued)

Term	Definition
Events	Events are representations of transitions between states. To this end, they consist of a number of *guards*, which represents pre-conditions for the event to fire, in addition to *parameters*, which are symbolic representations of values involved in the transition. Finally, there are a number of *actions* which are able to manipulate the state of the machine, primarily through acting on variables of the machine.
Axioms	Axioms declare properties of carrier sets and constants. They can therefore be used to provide type information for constants and carrier sets, in addition to specifying properties of carrier sets and constants, such as whether a carrier set is finite.
Theorems	Theorems can be utilised within both machines and contexts. They represent properties that must be derived from the axioms and other properties of the model. Every theorem will therefore generate a proof obligation which must be discharged in order to prove the theorem.
Guards	Guards are pre-conditions that must be true in order for an event to be enabled (i.e. the set of all guards for an event must equate to true before that event can occur). Guards are also utilised to provide type information to parameters within events.
Parameters	Parameters are symbolic representation of values that are relevant to a given event.
Variables	Variables are representations of the state of a machine. They are provided with type information through *invariants* and must be provided with an initial value within the *INITIALISATION* event of the model.
Invariant	Invariants are used to specify the type of variables. Additionally, invariants are also used to specify permitted values of variables, and therefore is used to constrain the valid *states* that the modelled system may be in, as the system state is represented by the machine's variables. All invariants must equate to true after each event, and proof obligations are generated on events in order to ensure that actions taken within events do not contradict invariants.

Table 3.1: Event-B terminology (continued)

Term	Definition
Actions	Actions occur within events and are used to manipulate the variables of the machine, while respecting the constraints represented by the invariants.

Proof obligations: The generation of *proof obligations* ensures that the Event-B model remains consistent and correct with regards to its axioms and invariants. The process of *discharging* a proof obligation is a formal demonstration (or proof) that the behaviour of some aspect of the model is consistent and as declared. Furthermore, a seeming inability to discharge a proof obligation is usually indicative that some aspect of the model has been incorrectly specified. Some major proof obligations that are frequently encountered in modelling systems using Event-B are as follows:

1. **INV**, the invariant preservation proof obligation, is generated on events that manipulate variables of the model. Discharging of this proof obligation ensures that events do not manipulate the variables of the model in a way that is contradictory with regards to invariants relevant to those variables.

2. **WD**, the well-definedness proof obligation, is generated to ensure that a potentially ill-defined expression of the model is in fact well-defined. Discharging this proof involves meeting a given 'well-definedness condition' of the expression.

3. **GRD**, the guard strengthening proof obligation, is generated to ensure that the concrete guards in a concrete event are stronger than the abstract guards in an abstract event. Discharging this proof ensures that, through having concrete guards strictly stronger than abstract guards, the concrete event being enabled means the underlying abstract event is also enabled.

Refinement: The process of refinement enables a formal model of a system in Event-B to be iteratively constructed such that more detail is added with each refinement. This therefore creates an ordered sequence of models in which either more functionality is added such as additional events, variables and invariants (known as *horizontal refinement*) or where the model and its variables become more precisely specified with the intent of the model being at sufficient detail to use it as a reference model for authoring of code or tests (known as *vertical refinement* or *data refinement*). Data refinements are a special case as they also require what are known as *gluing invariants* which map the specification of variables in the more concrete model to the variables in the abstract model that precedes it. This essentially 'glues', or formalises the relationship between, the refining machine and the refined machine.

3.7.3.2 Benefits of utilising Event-B within SE-STPA

The benefits of the chosen formal method are numerous:

1. The use of the Rodin tool for the Event-B formal method provides the ability to use interactive proving (and a variety of formal proof plug-ins) to demonstrate formally that the properties of the model hold or are unprovable in their current state. This can aid in improving the model (and thus the underlying system design) and also aid in improving the constraints themselves by increasing their specificity when attempting to ensure they meet the proof obligations of the model.
2. Model checking can provide counter-example traces where there is a possible series of events which may undermine any given constraints which can allow the analyst to consider development/refinements of each constraint to ensure that constraints cannot be undermined by a specific sequence of system behaviour.
3. The use of model checking within Rodin can additionally check that there are no deadlock states which may be a risk when constraining system behaviour. Due to the state-space involved in formal modelling of this type, model checking cannot totally guarantee that no deadlock states exist within the system.
4. Event-B supports the notion of refinement of a model as well as decomposing models into sub-models which can then be composed together once more to verify overall system behaviour. This can be of significant aid when carrying out the methodology in an iterative fashion over multiple system designs and then down into component-level analysis, as the formal model can be decomposed into sub-systems to keep pace with the analysis.

The last point is one of particular significance as it allows the formal method to keep pace with the system as its design becomes more detailed over time and can also be used in circumstances such as when a new subsystem is proposed for an existing system; the new subsystem can be analysed, modelled and confirmed to not breach any of the existing constraints.

The prescription of the Event-B model therefore serves to support SE-STPA by providing a higher degree of assurance that the constraints produced by SE-STPA sufficiently mitigate against undesirable system states and behaviours (as represented by hazards) through the ability to formally specify constraints on the system as invariants. In the standard STPA methodology, the constraints generated as a result of the analysis are not formally validated in any sense; though Leveson recommends the use of expert review in the analysis, this does not confirm the constraints are complete in terms of total, or even partial, mitigation against identified hazards. The methodology produced as a result of this thesis therefore aims to provide an additional verification of the robustness of its

constraints, when compared to STPA where constraints may only have any verification as to their completeness in mitigating against unsafe system states or behaviours once the system is being implemented or tested.

The integration of Event-B into SE-STPA creates two additional steps when compared to baseline STPA:

- The first step involves creating a formal model of the system prior to carrying out control action analysis/adversary modelling. This step represents an opportunity for an analyst to develop their understanding of the system and its documentation through having to translate it into a formal model. This may highlight any shortfalls in understanding how some of the controller interactions work, for instance.
- The second step involves the translation of constraints generated by adversary modelling and control action analysis into constraints on the formal model. This enables the constraints to be modelled to ensure they meaningfully address the issue they arose from, as well as ensuring that the set of all constraints do not unintentionally constrain the system into deadlock states or contradict each other. This improves the confidence that can be assigned to the set of constraints determined through analysis and reduces the risk that contradictory constraints may be integrated into the design simultaneously and not be uncovered until the later life-cycle stages.

3.7.4 Adjustment of terminology

In order to create a methodology capable of undertaking security and safety analysis as equal concerns, it is necessary to substitute some existing terminology from STPA in order to generalise the analysis such that it can be applied to both domains. While traditional STPA terminology has been used until this point of the thesis, subsequent chapters will make exclusive use of SE-STPA terminology.

3.7.4.1 *Accidents* vs *Losses*

One primary departure from the standard STPA is that SE-STPA substitutes the term of 'accident' for the more broadly applicable 'loss' due to the methodology aiming to identify and mitigate against both security and safety issues within the system design. The use of a term that is perhaps not entirely familiar or native to safety or security has been done to encourage analysts using the methodology to think in a multi-dimensional manner so as to avoid fixating purely on safety or security issues. Furthermore, the notion of 'accidents' has no real meaning in a security context, while 'vulnerabilities' works in a security context but is a foreign concept in safety analysis; thus, 'loss' has been selected as a word that is as broadly applicable as possible.

3.7.4.2 Critical Requirements

'Critical requirements' are SE-STPA's form of STPA's 'constraints'; this is once again attempting to avoid the anchoring effect with regards to terminology that is native to one domain while being foreign to the other domain. Utilising the terminology of '**critical** requirements' is done to attempt to highlight the importance of integrating the results of the analysis into not only the design but also the subsequent stages of the life cycle. Furthermore, as some of the resulting constraints produced by SE-STPA will cover what are traditionally described as 'functional requirements' (i.e. constraints which may modify system *behaviours*) while others will represent 'non-functional requirements' (i.e. constraints which may modify or define system *attributes*), it is essential that the terminology used does not conflict too significantly with standard systems engineering terminology which sometimes groups the notion of 'constraints' in with the non-functional requirements whereas 'critical requirements' may be of either category due to their status as previously unused terminology.

3.7.4.3 Hazards

Under the standard STPA methodology, the result of the analysis of each control action produces artefacts in the form of 'unsafe control actions'. Clearly, this terminology is not appropriate when the analysis is considering both safety and security issues. Therefore, SE-STPA substitutes this term such that the results of the control action analysis are also called 'hazards'. The justification for this is multi-faceted:

- The scenarios normally described by the 'unsafe control actions' artefacts resemble hazards in their wording and thus the additional category is superfluous.
- The 'unsafe control action' artefacts are generally then tagged with the identifier for a system-level hazard to which they are related, in order to create a pseudo-hierarchy.
- One or more critical requirements ('constraints' in standard STPA) are then attached to 'unsafe control actions' until the analyst is certain that the undesirable system states or behaviours are successfully mitigated against.
- The connection of sub-system level 'hazards' to system-level hazards can be of benefit when the system is decomposed and analysed one sub-system at a time as then the top level hazards for that analysis will be the sub-system level hazards. This can benefit assurance as there is then a hierarchy of hazards and mitigations, and identification of hazards that are not mitigated sufficiently will be immediately obvious when the hazards are drawn in the form of a tree diagram.

This substitution of terminology does not apply to adversary modelling. In adversary modelling, an adversary may have a number of action sequences which are manifestations of the adversary's intent, capabilities and access to the system. As a result of this, critical requirements are generated to address *action sequences* and so there is nothing termed as a 'hazard' during adversary modelling.

3.7.5 Modification of several steps that are carried over from the baseline STPA approach

The difference in steps between the baseline STPA approach and the SE-STPA approach are given in Table 3.2. Although several steps have been pulled through from the baseline approach, they have been modified, both in terms of the terminology utilised (covered in subsection 3.7.4) and additionally what each step actually involves from the analyst's perspective.

Table 3.2: Comparison of steps of STPA and SE-STPA

Step	STPA	SE-STPA
Step 1	Establish the system engineering basis	Establish the system engineering basis
Step 2	Draw a functional control structure for the system	Draw a functional control structure for the system
Step 3	Generate control actions	Generate control actions
Step 4	Identify unsafe control actions	Build the initial formal model
Step 5	Generate constraints to address unsafe control actions	Hazard analysis on control actions & critical requirement generation
Step 6	Perform scenario analysis/causal factors analysis	Adversarial modelling/analysis & critical requirement generation
Step 7	Iterate design or re-scope the analysis	Critical requirement integration into formal model
Step 8	*N/A*	Perform scenario analysis/causal factors analysis
Step 9	*N/A*	Iterate design or re-scope the analysis

Three of the steps carried forward from baseline STPA have undergone changes in SE-STPA:

1. Establish the system engineering basis.

2. Identify unsafe control actions.

3. Generate constraints to address unsafe control actions.

An examination of the changes to each step are considered in the sections following this one.

3.7.5.1 Establishing the system engineering basis

In the baseline STPA approach, this step involves the identification of accidents and hazards at the system level. *Accidents* are system states where the system has in some way failed to achieve its safety-related purpose. *Hazards* are system states which may, through a combination of environmental changes or further system activity, result in an accident. The STPA family of methodologies therefore focus substantially on identification and control of the system to ensure it does not enter a hazardous state, as hazards can eventually become accidents. A further consideration is defining the *purpose* statement of the system, from which all *accidents* are naturally derived, and which occurs prior to any derivation of system-level accidents or hazards.

In SE-STPA, the equivalent terminology is instead *losses* and *hazards*, respectively. This has been previously justified in subsubsection 3.7.4.1 but enables the analyst to think in both safety and security domains. This re-contextualisation is also intended with regards to the *purpose* statement that is generated when analysing a system; the system is likely to have a number of security goals in addition to safety goals, and so the purpose statement that is generated during the course of applying SE-STPA to the system shall likely be more extensive than it would when applying baseline STPA.

3.7.5.2 Identify unsafe control actions

This step has been expanded in terms of terminology, as covered in subsubsection 3.7.4.3, which enables safety and security issues to be teased out while undertaking analysis of control actions. The intent of this step in SE-STPA , where it is instead titled *Hazard analysis on control actions & critical requirement generation*, is to enable identification of both safety issues as well as security issues *relating to the control actions*. This step can therefore be considered to be focused primarily on safety issues, as well as security issues that can affect the primary means of maintaining safety (i.e. the control actions). Security issues that are not necessarily relating to maintaining safety, but rather consider the security of the platform as its own distinct aim, are instead considered in *Step 5 - Adversarial modelling & critical requirement generation* of SE-STPA .

3.7.5.3 Generate constraints to address unsafe control actions

This step has been pulled through and combined with the control action analysis step, so as to create the step of SE-STPA entitled *Hazard analysis on control actions & critical requirement generation.* The rationale for this combining is that Step 5 of SE-STPA also generates critical requirements, and so the critical requirements necessary to address hazardous control actions are best generated while the analysis on control actions is ongoing.

3.7.6 Reduction in dependence on expert review

As outlined in subsection 3.5.1, the baseline STPA technique requires both *system experts* as well as *domain experts* in order to ensure that the analysis is undertaken in the best possible manner, that the identified failure modes/threats are credible, and that the resulting constraints are therefore effective and provide adequate protection against the given failure modes/threats.

The benefit provided by SE-STPA is that, while system experts will still be required, the methodology itself can be utilised in a way that provides effective co-analysis and can compensate for a lack of available domain experts. This is due to it covering both safety & security in a systematic, structured way which enables thoughtfulness around how control can be undermined (degrading safety) and what adversaries/threat actors are likely to threaten the system as well as their likely attack paths/behaviours (degrading security). While this does not fully replace the need for domain experts, it can enable both security-by-design and safety-by-design to be achieved with traceable evidence.

Furthermore, if at a later date, domain expertise becomes available, the existing analysis and associated artefacts generated by SE-STPA can be considered by the domain expert and aspects of the analysis can be revisited if there are additional threats/hazards that are felt to be relevant to the system in its operational environment.

This approach represents a net improvement over the baseline STPA methodology as this primarily fixated on *safety* concerns, and was not underpinned by a theoretical model which accounted for the complex interactions between safety and security as emergent properties. By undertaking structured co-analysis, SE-STPA provides the capability to generate critical requirements to address perceived credible safety and security hazards with or without domain expertise being available.

3.8 Summary

This chapter intended to provide an overview of why the STAMP model and STPA analysis approach were chosen for modification to create the STAAMP model and the

SE-STPA analysis technique. It also details the modifications that have been made compared to the baseline model/analysis technique, and justifies the inclusion of the Event-B formal method in the methodology based on the capabilities that are offered by this formal method to enable verification of generated critical requirements to ensure they meaningfully address identified hazards of the system being modelled.

Chapter 4

SE-STPA in detail

4.1 Introduction

This chapter presents SE-STPA by exploring each step of the methodology through an illustrative example. This chapter is provided to detail the methodology prior to the detailed case study chapters, and therefore seeks to provide a high-level explanation of each step and its context.

4.2 A high-level overview of SE-STPA

4.2.1 Glossary

A glossary of key terms utilised throughout the methodology is provided in Table 4.1.

Table 4.1: Glossary of key SE-STPA terms

Key term	Description
Loss	A loss results from the system failing to achieve/meet its declared purpose. All losses are deemed to be of equal importance and are derived from the system purpose statement(s). A loss may represent a security breach or an accident; the term 'loss' is utilised due to its applicability in both security and safety domains.

Table 4.1: Glossary of key SE-STPA terms (continued)

Key term	Description
Hazard	Hazards are unsafe or insecure system states which, when coupled with worst-case environmental conditions, will result in a loss. STPA and its derived methodologies (including SE-STPA) aims to prevent losses by constraining the system and its behaviours such that it does not enter into hazardous states in the first place, as not all environmental conditions can be controlled and managed effectively.
Causal factor	Causal factors are behaviours/actions which contribute to a hazard. One example may be an operator being overloaded by poorly-designed alarms from a control system, another may be a failure/disruption in networking between components. The systems-theory based approach of STPA and SE-STPA take a holistic view of what may contribute to a hazard and so these may be from any aspect of the socio-technical spectrum.
Critical requirements	The terminology of critical requirements was chosen to represent what the standard STPA methodology terms as 'constraints'. In the baseline STPA approach, 'constraints' are generated on the system in order to prevent unsafe control actions. In SE-STPA, the use of 'critical requirements' embodies the concept of constraining the system, but terming these as 'requirements' has been done to emphasise that these are not just constraints that can be implemented and then ignored; critical requirements should be considered all throughout the system life-cycle as with any other type of requirements.

4.2.2 SE-STPA steps

SE-STPA consists of nine distinct steps, which are demonstrated visually in Figure 4.1. Each step is then further detailed in its own subsection.

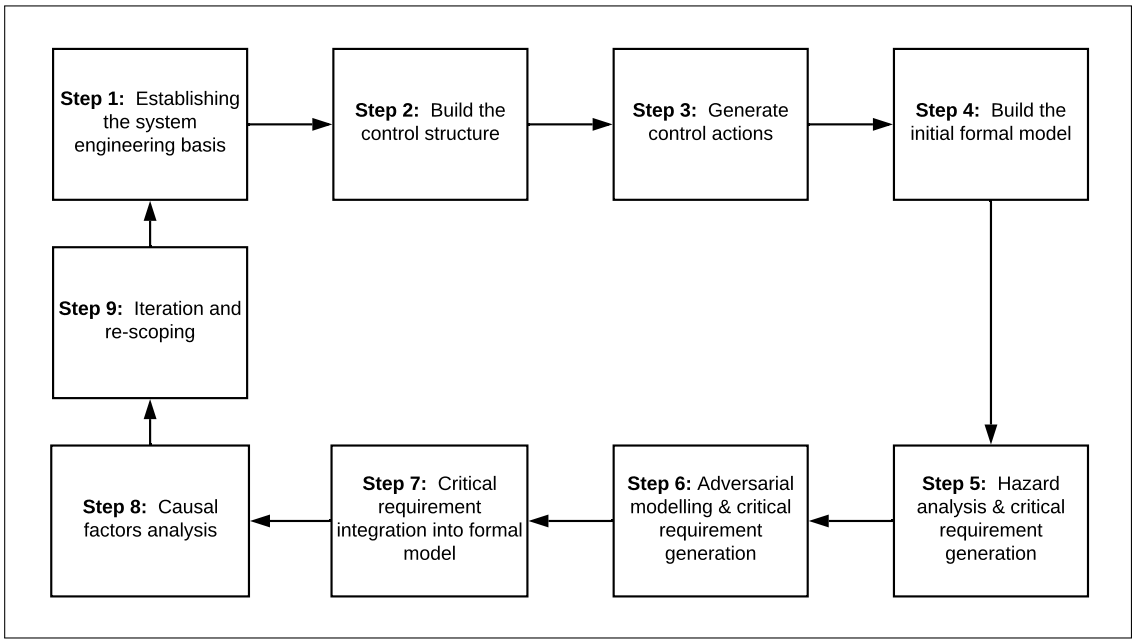

Figure 4.1: A visual summary of the steps of SE-STPA

4.2.3 Clarification on the illustrative example

The illustrative example used to highlight each step of the methodology within subsection 4.2.4 is based on an abstracted and simplified version of the end-to-end technical architecture of the UK's smart meter system [42]. The system therefore consists of a multitude of smart meters on consumers' premises, as well as a central system which they report to.

A case study was undertaken using this simplified smart meter architecture with an earlier form of SE-STPA; this is detailed, with lessons learned, in Chapter 5. This chapter therefore utilises the smart meter concept to illustrate each step of the methodology, but with the current form of the methodology instead, and so some steps within subsection 4.2.4 will not have an equivalent within Chapter 5.

4.2.4 Methodology steps

4.2.4.1 Step 1 - Establishing the system engineering basis

The analysis process begins by taking the system under analysis and defining its boundaries as well as the underlying purpose of the system. The underlying purpose of the system (or multiple purposes where the system may require this) and system boundaries should be explained in plain English, in the form of a statement.

In the context of the example, the purpose statement is:

> The system purpose is to maintain an accurate and up-to-date record of the electricity usage by a cluster of Meters and additionally, keep track of any issues arising from Meters failing to report their correct and current usage value in a timely manner. The system will additionally be responsible for keeping track of registered and retired meters, managing billing for each Meter, and for sending disconnect commands to Meters that have fallen behind on their billing.

The purpose statement(s) then allow the determination of a system's *losses* which essentially represent a failure of the system to carry out its *purpose*. These are essentially once more plain language statements representing an inability to meet any aspect of the purpose statement.

One of the *losses* associated with the example is therefore *"Loss of accurate registration/retired data for meters."* This loss represents a failure of the system to meet the second sentence within its purpose statement.

The final part of this step involves the identification of system-level *hazards* which are inferred from the *losses*. *Hazards* are essentially system states which may lead to losses and are therefore what the analysis seeks to develop mitigations against. As per the STAMP model, controllers are unlikely to have total control over the environment in which a system operates so preventing hazards from occurring becomes fundamental to ensuring that systems do not experience a loss, as losses are often just the result of a hazard occurring at the same time as some combination of negative environmental conditions.

An example of a hazard within the context of the example is *"Meter registration is not correctly recorded by the system"*. It is easiest to represent these in a table; for large collections of purposes, losses and hazards, it is more efficient to allocate identifiers to each item and then use tables to map identifiers alone. An example of a simple table format can be found in Table 4.2.

Table 4.2: Example purposes, hazards and losses mappings

Purpose	Loss	Hazard
P1: The system will be responsible for keeping track of registered and retired meters.	**L1:** Inaccurate registration/retirement data is held by the system.	**H1:** Meter registrations are not correctly recorded by the system. **H2:** Meter retirements are not actioned by the system.

System-level losses and hazards will also be mapped to by component-level hazards to create a hierarchy of possible ways of reaching a loss state. The analysis aims, in later

steps, to generate critical requirements and verify their effectiveness at mitigating against each potential 'path' of hazards to a loss. This aids the traceability of the analysis and provides assurance that the system-level issues are being considered when dealing with component- or control-level aspects.

Consideration of safety & security in relation to the purpose statement: It is worth noting that this step does not focus specifically on safety or security, but instead on how the system may fail to meet its purpose (a loss) and how system states may contribute to any given loss (a hazard). Due to the emergent nature of both safety and security, even if the purpose statement does not contain an explicit outline of the security aspects of the system, these will be considered thoroughly at later points in the analysis, with safety being considered primarily in the *hazard analysis step* (as detailed insubsubsection 4.2.4.5) and security being considered within the *adversary modelling step* (as detailed in subsubsection 4.2.4.6).

As these steps enable thoughtfulness around safety & security aspects of the system, as well as likely threat/hazard profiles, this can essentially compensate where the system purpose statement is more *functional* and has not provided much in the way of security or safety context. Furthermore, both of these steps work in a 'bottom-up' manner and so a lack of purpose or losses that explicitly capture possible safety/security concerns will not hinder the effectiveness of these steps.

4.2.4.2 Step 2 - Build the control structure

The construction of the functional control structure seeks to create a representation of all entities involved in the control of the system and any underlying processes with which the system interfaces. This may not necessarily map to individual components if the system design is moderately finalised; in fact, this step is designed such that it could be performed against a system that only consists of a set of requirements from which entities may be inferred.

Controllers (i.e. components that communicate with other components or control some underlying process) have both *responsibilities* (which can be viewed as a form of component *purpose* if this aids in visualisation) as well as *process models* which model the understanding that a controller has of aspects of the system state or underlying process that are relevant to it and its responsibilities. Controllers may also be passing commands around to one another and feedback may also be passed between either controllers and processes, or controllers and other controllers. It is important to not be too explicit in what exactly represents *commands* versus *feedback*, as this can be unduly restrictive. It may indeed also be the case that feedback and commands are transmitted in essentially the same way.

Whatever the state of the system in terms of how finalised the system design or architecture is, an abstract functional control structure should be developed from the available information on the system.

Going back to the smart meter example, an exemplar control structure for it is given in Figure 4.2. As can be seen, there are essentially two tiers of controllers, which pass commands/feedback between themselves. The *meter* controller is directly interfacing with the underlying process, which is the electricity supply.

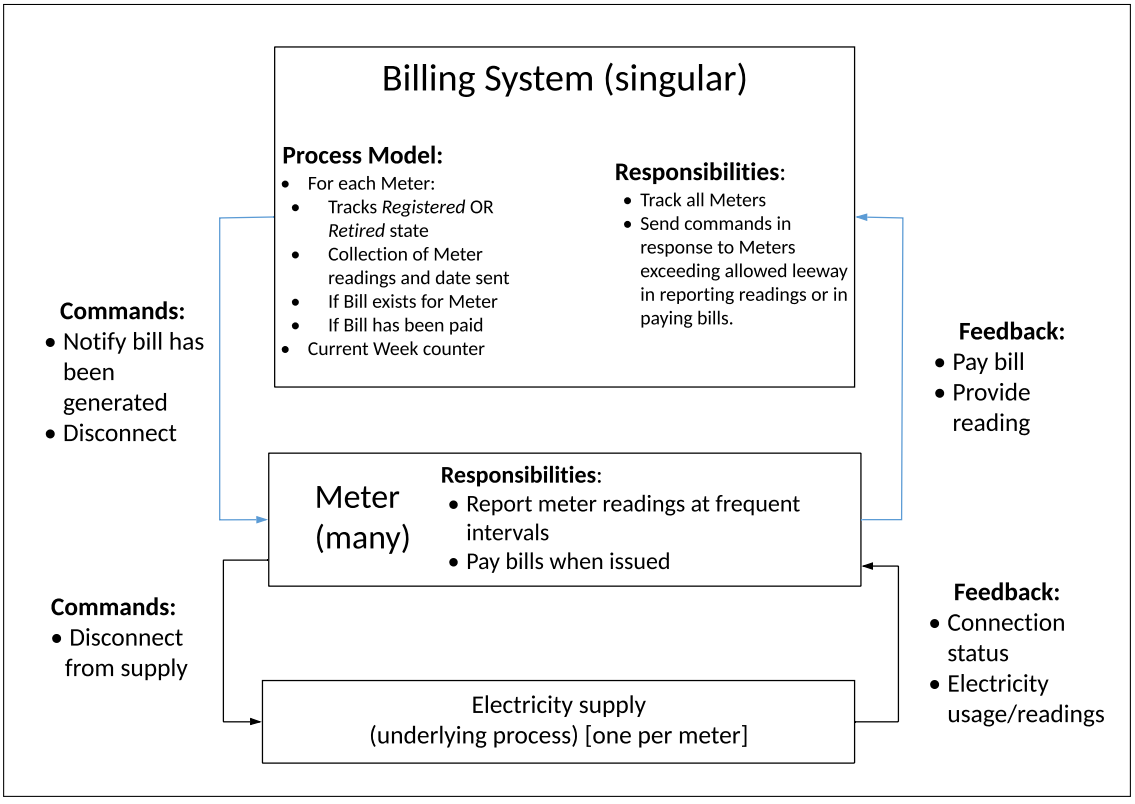

Figure 4.2: Functional control structure for smart meter example.

It can also be useful to express the cardinality of elements within the control structure rather than duplicating controllers/processes unnecessarily. This could be expressed through UML-like multiplicity, or alternatively, via annotations on each controller/process as has been used in the example.

4.2.4.3 Step 3 - Identify control actions

The development of the functional control structure should aid the identification of control actions; commands and feedback passed around the control loops are extremely likely to comprise the bulk of control actions available within the system. The purpose of this step is two-fold: to identify the control actions such that they can be analysed in

later steps is one element, but the identification also permits the creation of an initial formal model of the system in the next step.

Going back once more to the example, some clear control actions have arisen, such as:

- Register Meter.
- Retire Meter.
- Generate Bill.

Once all control actions have been identified, the next phase of the analysis can begin.

4.2.4.4 Step 4 - Building the initial formal model

At this stage, an initial formal model is constructed of the system using Event-B and its associated tool, Rodin. The purpose of this step is to allow for an abstraction of the system behaviours to be modelled (broadly in line with the functional control structure) and to ensure that all control actions that exist have been identified.

There is generally a straightforward transposition of control actions into one or more *events* within Event-B, while process models can be most easily modelled as combinations of *variables* and *invariants*. Restrictions or conditions on control actions can be modelled through *guards* on events. Constraints within the design are to be modelled on the system where these may already exist in requirements or are within the system design already; if control actions may only occur in certain circumstances, this should be modelled accurately.

The construction of the formal model can also aid in determining whether the understanding of the system is adequate; if it is unclear what effect a given control action may have on the receiving controller/process, or perhaps the process model seems incomplete, then this indicates that the system under analysis has been scoped incorrectly or that the functional control structure is incomplete.

Further detail on the initial formal model for the smart meter example can be found in subsection 5.3.4.

4.2.4.5 Step 5 - Control action analysis and identification of critical requirements

The control actions as determined in Step 3 (subsubsection 4.2.4.2) are then subjected to analysis through considering if any insecure/unsafe system states (or hazards) can occur if:

1. The control action is issued.
2. The control action is not issued.
3. The control action is issued too soon or too late within the expected sequence of system events.
4. The control action is continuous and is issued for too long/too short a period of time.

If there is the potential for a control action to cause an unsafe/insecure system state as a result of one of these conditions, the hazard is noted down (a tabular form is often used for this in STPA; with the four conditions as columns, and each control action as a row). All the hazards discovered through analysis of each control action can then be used to generate critical requirements which represent constraints or limits in natural language on when/how control actions may be issued; these critical requirements should seek to constrain the system such that the identified hazards may not arise.

Going back once more to the example, one can take the 'Register Meter' control action and subject it to analysis as is shown in Table 4.3 .

Table 4.3: Control action analysis results

Control Action	*Is issued*	*Is not issued*	*Is issued out of sequence*	*Is issued for incorrect duration*
Register Meter	An invalid meter is re-registered.	A meter fails to be registered.	A meter is registered multiple times.	N/A - registration is discrete.

The first output from this step is the information in each row which represents a set of *hazards* associated with each control action. Many of these will map back to the identified system-level hazards; this is intentional. Due to the fact that STPA works on a model of inadequate control, critical requirements are not directly generated for system-level hazards as these would essentially be vague and not beneficial to future stages of the system life-cycle. Instead, identification of all ways that hazardous control can produce contributory hazards towards system-level hazards is undertaken, and hazardous control is mitigated through the generation of critical requirements.

The second output from this step involves generating critical requirements to address the identified hazards. The generation of *critical requirements* is once again centered around natural-language statements that seek to address a given hazard. The critical requirements should be specific enough to address the hazard directly but without being

too prescriptive in terms of actual implementation. Some example hazards and critical requirements are given in Table 4.4.

Table 4.4: Critical requirement generation (control action analysis) from the example

Hazard	*Generated critical requirement*
H3: An invalid meter is re-registered.	**CR1:** Meters, once retired, may not be returned to a state of registration.
H4: A meter is registered multiple times.	**CR2:** Registration for a given meter may only occur once.

These critical requirements will be integrated into the formal model in a later step.

4.2.4.6 Step 6 - Adversary modelling and generation of further critical requirements

The next aspect of the analysis involves the notion of *adversary modelling*. An *adversary* is essentially an abstraction of any unauthorised party interacting with the system in a way that might undermine the system's purpose, including any implicit security requirements a system may have.

A given adversary consists of the following properties:

1. An identifier, such that other aspects of the analysis may be mapped back to this adversary.
2. A name/categorisation.
3. The *intent* of the adversary, ranging from something as minor as 'curiosity' all the way to 'denial of service' or 'permanent damage'.
4. The *access* or *perspective* of the adversary, which will ideally consist of some number of *manipulation points* - described later in this section.
5. The *information* held by the adversary, which may be expressed explicitly or more generically in terms on a scale such as 'minimal' to 'complete'.
6. The *actions* that an adversary may undertake. This may be a linear flowchart but may also consist of multiple paths of action that the adversary may carry out. These may be described in broad strokes, much like the identification of *hazards* from control actions as detailed in subsubsection 4.2.4.5.

The system's functional control structure itself must also be annotated with *manipulation points (MP)*; essentially, these are any communication links between controllers, or the controllers and processes themselves, that may feasibly be accessible to *any* adversary. Not all adversaries will utilise all manipulation points, but all interfaces between different entities in the functional control structure should be tagged with an identifier in order to facilitate the adversary modelling process.

The purpose of the *adversary modelling* is essentially to provide a traceable and explicit *security assurance case*: an artefact demonstrating that a given type of adversary has been considered, and has had sufficient mitigations against their behaviours generated in the form of critical requirements. Consideration of all possible actions that an adversary may take is not necessary; a sample can be provided to exemplify the capabilities of the attacker and allow critical requirements to be generated accordingly, if expert review determines that the most likely or damaging attack paths through the system have been considered[1].

Going back to the example, the annotated version of the functional control structure for the smart meter system is given in Figure 4.3, as well as a table containing the definition of an adversary in Table 4.5.

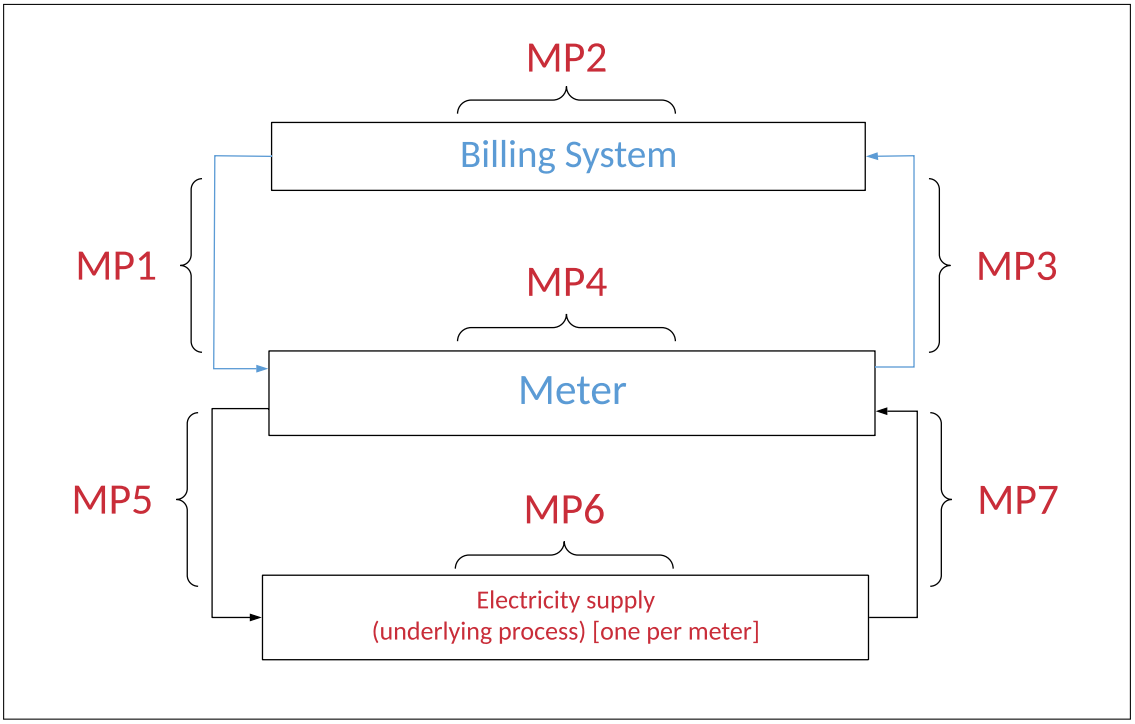

Figure 4.3: Annotated functional control structure for smart meter, with manipulation points.

[1]Equally, consideration may be focused around three model adversaries who represent an intent to undermine each aspect of the CIA triad and this could be utilised to demonstrate that confidentiality, integrity and availability of the system have been considered and addressed appropriately through critical requirements.

Table 4.5: Adversary description from the smart meter example

Property	*Detail*
Identifier	Adversary 1 (A1)
Name/categorisation	Fraudulent user
Intent	Adversary intends to reduce their bill through tampering with the meter.
Access	Adversary can see and manipulate MP5 and MP7.
Information	Adversary is aware that feedback channel MP7 reports electricity usage by the consumer.
Actions	Single step: Adversary intercepts and modifies MP7 result at all times. This has the net effect of putting the electricity consumption, as recorded by the smart meter, below what has actually been used.

As detailed briefly in subsubsection 3.7.4.3, *adversary actions* are viewed as analogous to *hazards* as adversaries and their actions cannot be directly controlled. Instead, much like hazards, the actions of an adversary are a result of a combination of factors that manifest themselves as a hazard; the adversary essentially can just be seen as a *malicious* environment. The intent is therefore to generate *critical requirements* as with any other type of hazard in order to ensure that control over the system can be maintained in a meaningful way in spite of the identified hazard. This differs from the later causal factors analysis, which is focused on the *derivation of possible contributing factors to a loss of control* to ensure that identified factors can be mitigated in the design or processes around the management of the system; adversaries seek to *directly undermine* system control with a malicious intent and therefore performing it at this step is a more meaningful way of modelling the true way that adversarial behaviour works rather than to simply consider it as a causal factor.

To generate critical requirements, a summary of the adversary behaviours can be used in lieu of the hazard column used in Table 4.4 and the table can otherwise be reused. For hazards that have originated from the adversary modelling, it can be simpler to summarise in terms of the functional control structure than discussing manipulation points, though occasionally the use of manipulation points in the description may bring greater clarity, so this is situational. Going back once more to the example, some hazards and their associated critical requirements are presented in Table 4.6.

Table 4.6: Critical requirement generation (adversary modelling) from the example

Hazard	*Generated critical requirement*
H5: Adversary manipulates data between electricity supply and meter to reduce reported electricity usage.	**CR4:** Meter will receive local average usage and will raise an alert if readings are more than 25% below this in a month's period. **CR5:** Billing System flags any meters that send alerts or provide readings that are more than 20% below the projections for that meter over a month's period.

Once a set of critical requirements have been collected through this generation process, an analyst would ordinarily proceed to the next step. However, if the critical requirements generated propose a significant modification of the design of the system - such as the addition of new entities to the control structure - then it may be worthwhile to iterate the design/requirements and begin the process once more from Step 1. The example has identified new commands between the two controllers in the form of the 'flagging' and the 'alert' control actions that are implicitly defined by the generated critical requirements and so this could inform a change to the functional control structure, which would necessitate beginning analysis again.

4.2.4.7 Step 7 - Integration of critical requirements into the formal model

The integration of the critical requirements leverages the existing initial formal model, represented in Event-B using the Rodin tool, to validate each critical requirement. Each *critical requirement* is integrated into the model in its own distinct *refinement* of the model in order to enable greater traceability.

The integration may consist of the following aspects which seek to extend and refine the model:

- Addition of invariants to the machine element of the model in order to constrain variables. An example below is an invariant indicating that the variables of registered meters and retired meters may not overlap:

$$\textit{MetersRetireOverlapInvariant:}$$
$$RegisteredMeters \cap RetiredMeters = \varnothing$$

- Addition of guards to events to narrow the circumstances in which they may occur. The RegisterMeter event can be restricted through the guard below to ensure that

retired meters may not be re-registered, for instance:

$$\textit{NotAlreadyRetiredGuard:}$$
$$meter \notin RetiredMeters$$

where *meter* is a parameter of type METER and *RetiredMeters* is the set of all retired meters.

- Addition of more actions to events. As part of the illustrative example, it was determined that the use of 'tokens' were a sufficiently capable abstraction of the notion of encryption between a meter and the billing system. Tokens represent a secret assigned to each meter during registration of a smart meter, and are required to be submitted to the billing system in order to take an action. It is therefore necessary to store a valid token when registering a meter as demonstrated by the following action:

$$\textit{tokenAssignAction:}$$
$$RegisteredTokens(meter) := token$$

- Addition of axioms to the context element of the model to add properties to the constants and carrier sets represented therein. The illustrative example did not leverage any axioms but one of the most commonly used axioms is an axiom to partition a given set such as a MeterType set being partitioned into two for electricity and gas meter types:

$$\textit{MeterTypePartitionAxiom:}$$
$$partition(\text{METERTYPE}, \{Electricity\}, \{Gas\})$$

- Addition of new events, variables and other elements to the model and restricting existing variables, events, etc. The example created new variables to model the token concept by storing all tokens seen by the system and all currently registered tokens:

$$\text{VARIABLES:}$$
$$AllTokens$$
$$RegisteredTokens$$

Additionally, the example made extensive use of event extension in refinements of the initial model to allow for additional events to be reused but have further guards and actions added to them.

The goal of the integration of critical requirements is to ensure that they mitigate against their associated hazard sufficiently. Many critical requirements will result in *invariants* being created and violation of these will be indicated by the tool through an inability to discharge all of the proof obligations associated with that invariant or alternatively through the identification of counterexamples through model checking.

The strength of the formal method in this regard is that one has both mathematical - i.e. proof obligations either being discharged or remaining undischarged - and operational - i.e. counter-examples being generated during model simulations - demonstrations that a critical requirement either prevents the system from reaching the hazard state or is unable to prevent the system from entering the hazard state. These can guide either further iteration on the critical requirements to ensure they do fully mitigate against the system entering the hazardous state through becoming more descriptive/detailed, or the iteration of the design itself to ensure critical requirements can be generated to sufficiently mitigate hazards.

4.2.4.8 Step 8 - Causal factors analysis

This step seeks to address *how* many of the identified hazardous behaviours in the system have arisen. This is less of a concern with the hazards identified as a result of *adversary modelling* as security is often framed as being an issue of "intentional actions by malevolent actors" [147], but is instead a more significant and meaningful task when undertaken against the hazards found through control action analysis, as safety concerns and analysis focus on preventing "unintentional actions by benelovent actors" [147] and therefore these hazards sometimes lack much in the way of context or causality. This stage of the analysis attempts to investigate this aspect.

The core process for undertaking this step of analysis should involve selecting each hazard in turn and subjecting it to *causal factors analysis* through considering how a series of actions or contexts in the control loop could lead to the hazard arising. A guidance diagram for this can be found in a variety of STPA sources which give guidance on exactly what may be considered a 'causal factor'. The analyst then attempts to generate further critical requirements or design changes in order to mitigate against any causal factors that are deemed to be reasonable. Each 'causal factor' can therefore essentially be viewed as a contributing sub-hazard to a larger hazard; examples of causal factors tend to be around ways in which sensors may misreport information, how communications can be disrupted, or how controller understanding can fall out of sync with the process they are modelling with their process models.

The importance of this step is that it should provide an opportunity for identifying how the design may not be optimal for ensuring control can be maintained of the underlying process or between control entities within the system design, and how this degradation

in control can eventually lead to a hazard. This allows aspects of hazardous control that are commonly overlooked, such as human factors in terms of interfaces or understanding of feedback, to be considered in a meaningful way and to be treated as first-rate concerns within future system designs and iterations.

Going back once more to the example, the hazard of 'A retired meter is re-registered' can be subjected to causal factors analysis; this can be seen in Figure 4.4. The controllers in this example have been annotated with what causal factors might cause the hazard as well as any commands/feedback that may be passed that may also lead to the hazard. The causal factors are then addressed in Table 4.7.

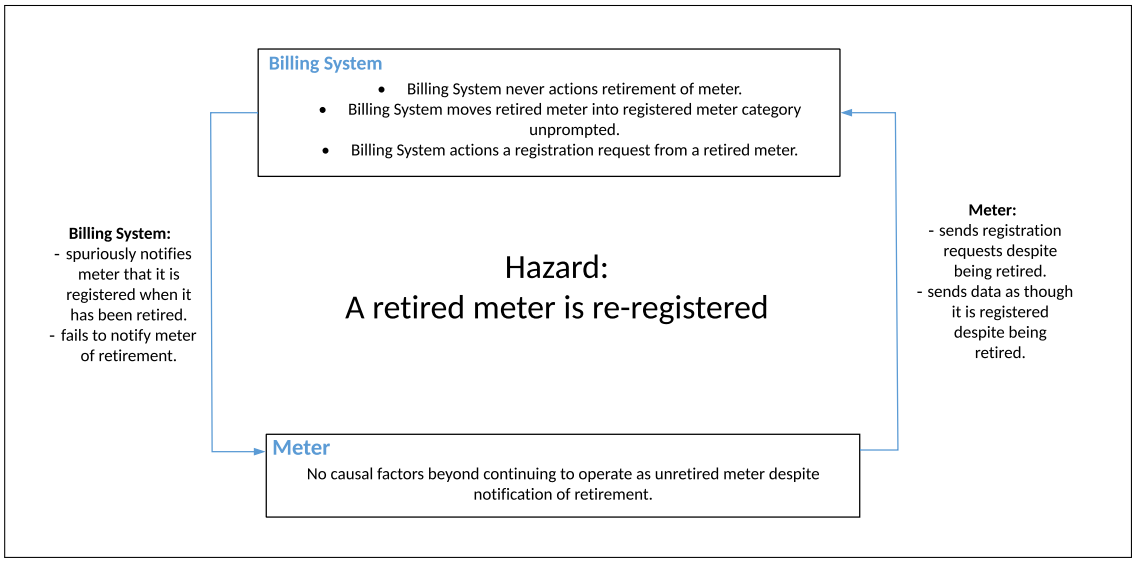

Figure 4.4: Causal factor analysis of one hazard from the example.

Table 4.7: Causal factors and the resulting critical requirements/design changes

Causal factor	*Critical requirement or design change*
H6: Meters continue to send data and take no notice of their retirement which may cause the billing system to erroneously respond to commands.	**CR6:** The Billing System will ignore all communications from meters once they have been retired.

As a further example of causal analysis, if a human operator was included to serve as a controller that sits hierarchically above the Billing System, it may perhaps be that meters that are correctly reporting readings and appear in every way to be active (despite being retired previously) may be assumed to have been mistakenly retired, which can cause a retired meter to be re-registered erroneously. The causal factors analysis can then reveal

how the human operator (or indeed, any controller) can come to such a conclusion, and ensure that design modifications and constraints are generated to ensure this does not happen.

4.2.4.9 Step 9 - Iteration and re-scoping

The final step of analysis, shared with the baseline STPA analysis process, is the notion of iterating the design, and re-performing the analysis. This is due to the fact that some of the constraints generated by the baseline STPA methodology may recommend architectural changes, fundamental shifts in the process models of controllers, the addition of new control actions or indeed the addition of brand new entities to the control structure of the system. Another possible end-state of one pass of analysis is that the system requires only minimal changes in terms of design, and therefore the existing design is taken forward with a small number of modifications plus the set of constraints that have been generated as part of the analysis. These constraints will aid the subsequent steps in the system life-cycle and so do not necessarily have to be fully integrated into the design, and they may indeed represent behaviours the system should have once made more concrete.

In SE-STPA, a similar view is taken; the result of the analysis may be a set of *critical requirements* that propose sweeping changes to the architecture of the system and the entities within, or the critical requirements may simply refine and curtail certain undesirable behaviours and system states which can lead to hazards. The end-goal is to ensure that the system maintains security and safety through complete and thorough enforcement of critical requirements to ensure the system is not able to drift towards the boundaries of safety and security as detailed in subsection 3.7.1. If the design has potentially hazardous aspects that are difficult to remediate without a re-design, then this should be undertaken prior to further analysis.

4.3 Tool support (Rodin, Lucidchart, etc)

The tools used to support the analysis are also necessary to describe due to the methodology's aim to be "tool-supported". The primary tool used to support analysis is the Rodin toolset [2], but other tools have been used to support the visualisation of the system and the drawing of control diagrams etc. The primary tool used for this purpose is Lucidchart [95] which is a cloud-based diagramming tool.

4.3.1 Rodin

Rodin is a toolset and modelling environment designed for use with the Event-B formal method [3, 2]. It allows for models to be constructed through a user-interface, as well as providing interfaces enabling users to consider and discharge proof obligations that exist on the model through manual proving as well as automated provers. Rodin also allows for model checking to be undertaken through its integration of ProB; ProB enables models to be simulated interactively, allowing users to step through a model event-by-event, as well as permitting automated checks to enable detection of deadlocks and invariant violations [83, 84].

Rodin is highly extensible due to being built using the open-source Eclipse platform. This permits the development of plugins which can provide new functionality from improved text editing capabilities to allowing new provers to be integrated into the platform [67].

Many of the capabilities of the basic Rodin platform were leveraged as part of this thesis and in support of SE-STPA. The *proving* perspective within Rodin was utilised to discharge proof obligations on case studies, the *Event-B* perspective was used for editing and refining the formal model, and the *ProB* perspective allowed the model to be stepped-through which permitted model behaviour to be explored in real time.

The platform was also augmented by several plug-ins to enable other activities as part of this thesis:

- The *Event-B to LaTeX exporter* plugin was used to export Event-B models to a LaTeX-compatible format such that models could be included in appendices of this document where relevant to case studies.
- The *SMT Solvers* plugin was used to enable a larger number of proof obligations to be discharged automatically by providing more automated solvers for use within Rodin. This enabled more time to be spent on proving more complex proof obligations associated with case studies.

The tool support provided by Rodin allowed for models to be rapidly refined, and therefore enabled the methodology through providing appropriate and capable tooling which encouraged the usage of the formal method within several steps of the methodology. The net result of this was a higher degree of assurance that generated critical requirements were robust and addressed the hazardous behaviours that they sought to mitigate against.

4.3.2 Lucidchart

Lucidchart is an online, cloud-based diagramming tool that allows for free-form diagramming, as well as providing a variety of export file formats [95]. The use of Lucidchart within this thesis was primarily in creating the variety of figures throughout this document such as functional control structures for the case studies, diagrams outlining the high-level steps of the methodology, etc. These visual aids help both the understanding of the methodology, as well as enabling the presentation of aspects of case studies in a visual format. The latter point is particularly important as the drawing of functional control structures is core to several steps of Security-Enhanced Systems-Theoretic Process Analysis.

Lucidchart as a platform has a variety of collaborative features and integrations above-and-beyond what was used as part of the production of this document, and can interoperate with a variety of desktop software that performs similar functions.

Chapter 5

Smart meter case study

5.1 Introduction

This chapter details the application of a previous version of SE-STPA to a smart meter case study. It is presented as the case study was undertaken. A section on the lessons learned from this case study and the impact that was had on the methodology can be found in Section 5.4.

5.2 A summary of the case study

The case study presented within this case study is based on an abstracted and simplified version of the end-to-end technical architecture of the UK's smart meter system [42].

The system under consideration will be a 'smart meter' and its infrastructure, wherein electricity metering is performed by a low-powered cyber-physical system which has a responsibility to periodically transmit the current meter reading of a consumer's electricity usage to a remote system which then executes billing and manages payment for generated bills, as well as disconnecting a consumer's electricity supply.

The system is therefore comprised of two substantial subsystems – the remote end and the meter itself. The remote end is assumed to be connected to a multitude of meters at any given time and therefore manages at least two meters at any given time. The remote end is described as though it is located somewhere on the supplier premises and it can only receive information from meters through whatever network link exists between them. The meter itself is viewed as being only active during its time on consumer premises, where it is referred to as 'registered'. A meter which ceases to be 'registered' will be 'retired' but may not return from a state of retirement as this represents the meter being decommissioned and replaced by another meter on the customer premises.

Meters may also be 'disconnected' as a third possible status and this may only occur with meters which are 'registered' and a meter may then be subsequently retired at any point after a disconnection; this would represent a state in which a meter is suspected of either being damaged or compromised and will be commanded to temporarily disconnect the supply to the consumer premises.

For the ease of description, the system components shall be referred to as follows:

Billing System: responsible for communicating with meters, keeping track of the billing state of the accounts and validating that correct meter readings are being submitted in good time.

Meter(s): responsible for making measurements and submitting them to the Billing System, as well as submitting bill payments.

For the ease of analysis, Consumers and Meters are to be viewed as synonymous such that a Meter is required to perform actions that might be traditionally performed outside of the interactions between Meter and Billing System; this includes submitting bill payments which would traditionally form another control loop.

The Billing System's primary role is to maintain accurate billing and payment information for each meter connected. To maintain this, the Billing System may keep internal state information about the last billed and last paid information for each Meter. The Billing System is also authorised to send disconnect signals to Meters which have failed to send readings for 4 weeks or have failed to pay a bill in 8 weeks. Meters are allowed 12 weeks of 'grace' before either of these restrictions will be actively checked for that Meter. This is to permit post-installation issues with the smart meter to be diagnosed without an effect on the end-user.

5.3 Application of the methodology

5.3.1 Step 1 - Establish the system engineering basis

As part of scoping the system under analysis, it is clear from the documentation that the Billing System is the core of the system and will therefore be the centre of the analysis. The Billing System communicates with Meters but there is not a fully trusting relationship as the Billing System will maintain additional data on each Meter to determine if it is still reporting faithfully and often enough.

The system purpose can therefore be succinctly summarised as the following:

> The system purpose is to maintain an accurate and up-to-date record of the electricity usage by a cluster of Meters and additionally keep track of any issues arising from Meters failing to report their correct and current usage value in a timely manner. The system will additionally be responsible for keeping track of registered and retired meters, managing billing for each Meter, and for sending disconnect commands to Meters that have fallen behind on their billing.

This can be dissected into four inter-related (but distinct) purposes (to aid in representation and classification of each purpose):

Purpose 1: Keeping track of the registered/retired/disconnected state of all Meters known to the system.

Purpose 2: Accurate tracking of Meter readings and the frequency with which Meters send their readings.

Purpose 3: Generation and monitoring of billing for meters.

Purpose 4: Sending of disconnect commands to Meters that have failed to pay bills for a unacceptably long period of time.

Of the above, purposes 1, 2 and 3 clearly have *security* concerns associated, as information *integrity* will be compromised, while purpose 4 possesses a *safety* concern due to the fact that the disabling of a Meter can cause issues if a given consumer relies on electricity to provide heating or has medical need of electrical power to power medical equipment.

In order to determine a loss state, each purpose is taken under consideration for ways in which the system may ultimately fail to meet this purpose (from either a safety or security perspective). Each loss will be attached to one or more purpose statements via its identifier as shown in Table 5.1.

Table 5.1: Loss identifiers, descriptions and mapping to purposes

Loss identifier	Description	Purposes
Loss 1 (L.1.)	Loss of accurate registration/retired data for meters.	*P.1.*, *P.2.*[1], *P.3.*[2]
Loss 2 (L.2.)	Loss of ability to correctly maintain connection/disconnection status for all Meters.	*P.4.*
Loss 3 (L.3.)	Loss of ability to generate valid and correct bills for Meters.	*P.3.*
Loss 4 (L.4.)	Loss of ability to track Meter reading accuracy and timeliness.	*P.2.*, *P.3.*[3], *P.4.*[4]

These losses are then utilised to generate system-level hazards; these may be fairly generic to begin as they represent top-level system states; later steps will generate much more specific hazards but these must be done initially. These can be found in Table 5.2.

Table 5.2: Hazard identifiers, descriptions and mappings to losses

Hazard identifier	Description	Losses
Hazard 1 (H.1.)	Meter registration is not correctly recorded by the system.	*L.1.*, *L.4.*, *L.3.*
Hazard 2 (H.2.)	Meters fail to provide correct reading information.	*L.3.*, *L.4.*
Hazard 3 (H.3.)	Meters are incorrectly treated as retired while they should not be.	*L.1.*, *L.3.*, *L.4.*
Hazard 4 (H.4.)	Meters are disconnected when they should not be.	*L.2.*
Hazard 5 (H.5.)	Meters remain connected when they should have been disconnected.	*L.2.*

[1] Inability to track Meter registration/retired status precludes the ability to accurately track Meter readings; if the system is unaware of a Meter that **should** be providing readings then it must have additionally failed to keep accurate readings for that Meter.

[2] As before, correct billing first requires the system to know of all of its Meters.

[3] Correct billing is dependent on accurate reading information for each Meter.

[4] Not being able to to keep track of how accurate/timely Meter readings are (perhaps due to replay attacks - or inaccurate readings - as two examples) means that Meters which should be disconnected may not be disconnected, etc. and so this crosses over to Purpose 4.

Table 5.2: Hazard identifiers, descriptions and mappings to losses (continued)

Hazard identifier	**Description**	**Losses**
Hazard 6 (H.6.)	Meters fail to pay bills and correct action is not taken.	*L.2., L.3.*

With purposes, losses and hazards broadly identified at a system-level and based upon the provided description of the system under analysis, it is then possible to advance to Step 2.

5.3.2 Step 2 - Build the control structure

The information provided in subsection 5.3.1 has made this step significantly easier through identification what the important entities involved in the system are; this information would ordinarily have to be derived from any existing designs of the system or proposed based off of elicitation of requirements.

From this information, it is then possible to develop a detailed functional control structure, enumerating exactly what commands or data may pass between entities of the control structure as well as what each entity in the control structure keeps track of in terms of state information; this is also known as a 'process model'. This final functional control structure can be seen in Figure 5.1.

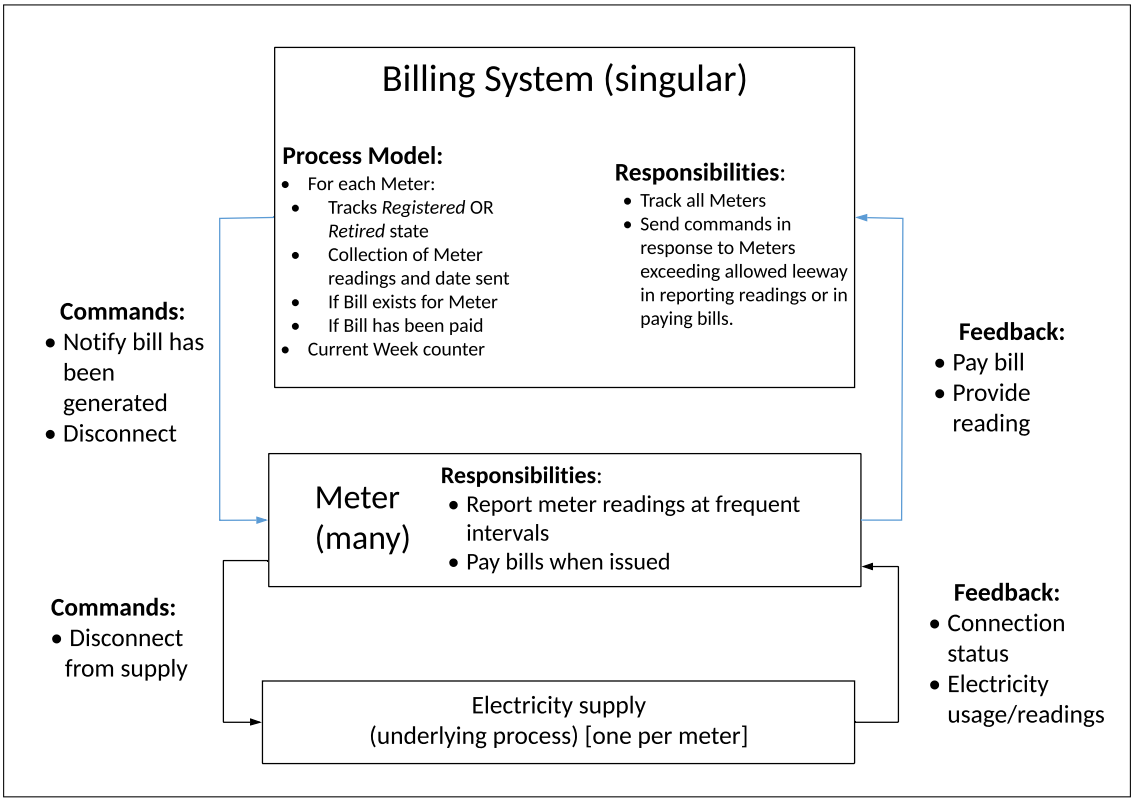

Figure 5.1: Final functional control structure

It is clear within this figure that many of the 'internal' commands make up the process model[5] for the Billing System. Furthermore, as the Meter does not need to track much in the way of internal state, it does not possess a process model within this analysis. The 'commands' and 'feedback' channels hint towards potential control actions that may be of use in the next step.

5.3.3 Step 3 - Generate control actions

At this stage, all possible control actions are enumerated in order to cover all actions[6] which may be taken within the boundaries of the system, following the process detailed in subsubsection 4.2.4.3 and derived from Figure 5.1:

1. Register Meter.
2. Retire Meter.
3. Submit Meter Reading.

[5] The process model as defined in STPA/STAMP refers to the information that a controller maintains about the process it is monitoring and acting on.

[6] As the system description in subsection 5.3.1 does not declare a workflow for the reconnection of a meter, the author decided to not consider this as a control action as it appears to not be in the scope of the system.

4. Generate Bill.
5. Pay Bill.
6. Disconnect Supply.

5.3.4 Step 4 - Build the initial formal model

The construction of the formal model involves the translation of the previously identified control actions and other system elements into an Event-B formal specification utilising the Rodin toolset.

This is done through the modelling of each control action as an event (where possible) and the representation of elements of the process model as variables and invariants. Additional events and variables have to be added in order to model some natural phenomenon such as the passage of time.

The initial formal model provides a foundation for the later steps and enables thoughtfulness around the system as specified. It can allow for some theorem proving and model checking to ensure that the model is free of deadlocks which may be caused by too many contradictory requirements on the design, as well as providing proof that the formal model has sensible events and variables through the satisfaction of the proof obligations. Furthermore, the formal model enables the analyst to consider whether the system behaviours are adequately specified in order to facilitate a model of the system, and if not, this can allow the system specification to be refined prior to the rest of the analysis being started.

5.3.4.1 Outline of the formal model

In this section, an overview of the formal model as used as part of this case study is provided. It will outline the major variables and events and how these relate to the preceding steps of the analysis. A full listing of the initial formal model can be found in Appendix A.

Based off the results from Step 3, the initial formal model consisted of the following events:

1. **INITIALISATION**, a default event required by Event-B, used for initialisation of all variables to some valid default state.
2. **RegisterMeter**, enabling the registration of a smart meter through the addition of the meter parameter to the *RegisteredMeters* variable.

3. **RetireMeter**, which marks a smart meter as retired by adding it to the *RetiredMeters* variable and removes it from variables associated with registered meters.
4. **GenerateBill**, which attaches a bill to a meter via the *MeterBill* variable.
5. **PayBill**, which marks an attached bill as 'paid' within the *BillPaid* variable.
6. **AddMeterReading**, which adds a new meter reading via the *MeterReading* and *ReadingValue* variables.
7. **AdvanceTime**, which represents the passage of time in terms of *weeks* and updates the *CurrentWeek* variable.
8. **DisconnectMeter**, which represents an irreversible change of state of the meter to *disconnected*, which manifests in the model in the *MeterDisconnectState* variable.

A mapping is provided in Table 5.3 to detail to how each of the control actions determined in Step 3 (subsection 5.3.3) maps to a representation within the formal model.

Table 5.3: Control actions mapped to formal event representations

Control action	**Formal event representation**
Not based on any control action; the INITIALISATION event is a mandatory component of any Event-B model	INITIALISATION
Register Meter	RegisterMeter
Retire Meter	RetireMeter
Submit Meter Reading	AddMeterReading
Generate Bill	GenerateBill
Pay Bill	PayBill
Disconnect Supply	DisconnectMeter
Not based on any control action; the system as detailed in subsection 5.3.1 involves the notion of time and so the AdvanceTime event has been created to simulate passage of real time	AdvanceTime

Additionally, the major variables represented by the initial formal model are therefore:

1. **RegisteredMeters**, a variable representing all currently registered meters; typed as a subset of the *METERS* carrier set, declared in the SmartMeterContext.

2. **RetiredMeters**, a variable representing all meters that have been marked as retired; a subset of the *METERS* carrier set.
3. **MeterReading**, a variable representing a mapping from meters to all recorded readings; typed as a relation between the *METERS* and *READINGS* carrier sets.
4. **BillPaid**, a variable representing whether a given bill has been paid; typed as a relation between the *BILLS* carrier set and *BOOL* type.
5. **CurrentWeek**, a variable representing the current week; typed as a natural number ($\mathbb{N}$).
6. **AllBills**, a variable representing all bills issued by the Billing System; typed as a subset of the *BILLS* carrier set.
7. **MeterBill**, a variable representing a mapping from all known meters (i.e. RegisteredMeters $\cup$ RetiredMeters) to the *AllBills* set. This represents the concept of which meter 'owns' which bill and is modelled as a surjective relation.
8. **MeterDisconnectState**, a variable representing whether a meter has been disconnected, which is irreversible within the context of the model; typed as a partial function between the *METERS* and *BILLS* carrier sets.

This model serves as the basis for the later integration of critical requirements.

5.3.5 Step 5 - Hazard analysis and critical requirement generation

Hazard analysis is then performed on each of the control actions in order to understand how hazards may occur in the context of any given control action. The control actions are analysed in line with subsubsection 4.2.4.5 and the figure covering each control action can be found in Table 5.4.

In Table 5.4, the *'Is issued for incorrect duration'* analysis condition has been included. This could be removed in the circumstance that the analyst is convinced all control actions are discrete and thus the notion of a 'duration' is irrelevant to the control actions being considered, as is the case here. It is left in merely for completeness.

An example of a continuous control action may be one that controls a motorised actuator or similar, where a control action must be constantly transmitted until the actuator has transitioned between states. Failure to transmit that control action for the correct duration may therefore leave the actuator in between desirable states and put the system into a hazardous state. A real-world example of such a control action would be the flaps on an aircraft wing; a controller may intend to extend the flaps to a given degree and this may require the control action to be provided until the appropriate degree of extension is reached. Were this interrupted, it may result in a hazard due to the flaps not reaching the desired state, which can have an undesirable effect on aircraft handling.

Table 5.4: Control action analysis results

Control Action	*Is issued*	*Is not issued*	*Is issued out of sequence*	*Is issued for incorrect duration*
Register Meter	A retired meter is re-registered.	A meter somehow fails to be registered.	A meter is registered multiple times.	N/A - registration is discrete.
Retire Meter	Retiring of a meter which should remain registered may have adverse effects.	Failing to retire a Meter which should be retired.	Retiring a meter which still has bills.	N/A - retirement is a discrete action.
Submit Meter Reading	Readings are submitted that are invalid. --- Readings are recorded from the wrong meter.	Readings are not submitted and billing becomes impossible.	Readings are submitted in the wrong sequence.	N/A - submitting readings is a discrete action.
Generate Bill	Bills that have already been paid are generated again.	Meters which should be issued bills are not.	Generating a bill too soon after a previous bill has been issued.	N/A - generate bill is a discrete action.
Pay Bill	A meter pays the bill of another meter.	None.	Attempting to pay a bill before a bill has been generated.	N/A - paying a bill is a discrete action.

Table 5.4: Control action analysis results (continued)

Control Action	*Is issued*	*Is not issued*	*Is issued out of sequence*	*Is issued for incorrect duration*
Disconnect Supply	The wrong meter is commanded to disconnect.	A meter is not commanded to disconnect when it should be.	A meter is commanded to disconnect prior to being registered. --- A retired meter is disconnected.	N/A - disconnecting a meter is a discrete action.

Hazards in Table 5.4 are identified by considering the way that a control action may become hazardous in any of the given cases (provided in the header of each column). This means a consideration of the control action under analysis in each of the cases while also attempting to cover all possible environmental or system states that the case may occur in. Taking an example from the table above, issuing the 'Register Meter' control action may become hazardous in the event the action *is issued* and the meter in question has already been marked as *retired*. This is clearly outside of the expected system behaviour, but the analysis is not restricted to respecting the usual flow of system events or states and therefore it can be ensured that all possible hazards can be found and have critical requirements generated so as to ensure the hazards do not arise.

These identified hazards may relate back to the previously identified system-level hazards as described in Table 5.2 but this does not necessarily have to constrain or guide the control action analysis; it may be that hazards are identified as a result of the analysis which do not connect innately with any previously identified system-level hazard. Nonetheless, it is anticipated that the majority of identified hazards will relate back to the system-level ones.

To continue the analysis, each of the cells in the table that are not marked as N/A, 'none' or are otherwise empty will be extracted in order to identify whether they are true hazards (and therefore in need of mitigation) or represent issues which are beyond the scope of the analysis. For those that are true hazards, critical requirements shall be generated for them. This can be seen in Table 5.5. In the interest of brevity, a table mapping these Hazards (determined through control action analysis) onto existing Hazards/Loss artefacts (determined at system-level) is not provided but this would be fairly straightforward as each of the hazards in Table 5.5 will fall under one of the previously-defined system-level hazards.

Table 5.5: Identification of hazards and critical requirement generation

Identified hazard	**Resulting Critical Requirement**
[H.7.] A retired meter is re-registered.	*[CR.1.]* Meters may not return from a state of retirement.
[H.8.] A meter somehow fails to be registered.	Out-of-scope of analysis - the assumption is that all known meters are registered.
[H.9.] A meter is registered multiple times.	*[CR.2.]* During registration, meters must be checked to ensure they are not registered twice.
[H.10.] Retirement of a meter which should remain registered may have adverse effects.	*[CR.3.]* Meters with outstanding bills may not be retired.
[H.11.] Failing to retire a Meter which should be retired.	Out-of-scope of analysis - meters are retired using a work-flow that is unspecified in the documentation.
[H.12.] Retiring a meter which still has bills	Duplicate of *[CR.3.]*
[H.13.] Readings are submitted that are invalid.	*[CR.4.]* Meters' readings must strictly increase.
[H.14.] Readings are recorded from the wrong meter.	*[CR.5.]* Meters must have some token that is known only to them and the billing system and must use this to authorise actions.
[H.15.] Readings are not submitted and billing becomes impossible.	*[CR.6.]* All meters should be checked as to whether they have fallen behind with submitting readings before the system's internal time may advance.
[H.16.] Readings are submitted in the wrong sequence.	Duplicate of *[CR.4.]*
[H.17.] Bills that have already been paid are generated again.	*[CR.7.]* Each bill must be unique.
[H.18.] Meters which should be issued bills are not.	*[CR.8.]* All meters which are due to have bills generated shall have these generated in a timely manner.
[H.19.] Generating a bill too soon after a previous bill has been issued.	*[CR.9.]* Billing should be carried out with a fixed frequency.

Table 5.5: Identification of hazards and critical requirement generation (continued)

Identified hazard	*Resulting Critical Requirement*
[H.20.] A meter pays the bill of another meter.	Duplicate of *[CR.5.]*
[H.21.] Attempting to pay a bill before a bill has been generated.	*[CR.10.]* Meters must specify which bill they are paying.
[H.22.] An incorrect meter is disconnected.	*[CR.11.]* Only meters matching the disconnection criteria may be disconnected.
[H.23.] A meter is not disconnected when it should be disconnected.	*[CR.12.]* A meter meeting the disconnection criteria must be disconnected in a timely manner.
[H.24.] A meter is disconnected prior to being registered.	*[CR.13.]* Meters may only be disconnected if they are in registered state.
[H.25.] A retired meter is disconnected.	Duplicate of *[CR.13]*

5.3.5.1 The notion of 'tokens' and its justification

The most significant of the critical requirements is **CR5** which mandates that all meters should have a unique token that is only known to them and the billing system, and that they must use this to authorise actions. This represents a fairly significant design change, and is intended to represent a generic critical requirement representation of the notion of security with the aim of preventing attackers or maliciously-modified meters from undertaking actions as though they were another meter. By providing a critical requirement defining the abstract notion of a token, a degree of flexibility is provided at later iterations where this may be concretely implemented using symmetric or asymmetric encryption, as an example.

5.3.6 Step 6 - Critical requirement integration

The integration of the critical requirements leverages the Rodin tool and the underlying formal representation in Event-B in order to constrain the model and to demonstrate the effectiveness of each critical requirement. In Table 5.6, a summary of how each critical requirement was integrated into a refinement of the existing formal model can be found. For the full model, this can be found without commentary in Appendix B.

The critical requirements were integrated into a single refinement of the system due to the relatively small size of the initial system model, but this would be unusual in a system of larger scope, as attempting to integrate all possible critical requirements into a single model refinement would represent an unwieldy and unnecessary challenge. The recommendation from subsubsection 4.2.4.7, which states that each critical requirement should be integrated into its own refinement, remains the recommended practice.

Table 5.6: Summary of integration steps for each critical requirement

Critical Requirement	**Adjustments made to model**
[CR.1.] Meters may not return from a state of retirement.	• Creation of Machine invariant (*MetersRetireOverlapInvariant*) to assert that the intersection of the RegisteredMeters and RetiredMeters variables should be the empty set. • Guard added to *RegisterMeter* event to constrain such that any meter parameter may not already be within the *RetiredMeters* set (*NotAlreadyRetiredGuard*).
[CR.2.] During registration, meters must be checked to ensure they are not registered twice.	Guard added to the *RegisterMeter* event to constrain such that a meter parameter may not already be within the *RegisteredMeters* set (*NotAlreadyRegisteredGuard*).
[CR.3.] Meters with outstanding bills may not be retired.	Guard added to the *RetireMeter* event to constrain such that a meter with any outstanding bills will not be eligible for being marked as retired (*RetireMeterMayNotHaveBillGuard*).
[CR.4.] Meters' readings must strictly increase.	Guard added to the *AddMeterReading* event to enforce that each new meter reading must be greater than the preceding value (*readingIncreasesGuard*).

Table 5.6: Summary of integration steps for each critical requirement (continued)

Critical Requirement	**Adjustments made to model**
[CR.5.] Meters must have some token that is known only to them and the billing system, and must be used to authorise actions.	• Creation of a *TOKENS* carrier set within the Context (*SmartMeterContextFull*). • Creation of an *AllTokens* variable (and associated invariant) representing a subset of the *TOKENS* carrier set to keep track of all used tokens during the lifetime of the system (*AllTokensTypeInvariant*). • Creation of an *RegisteredTokens* variable (and associated invariant) to map *RegisteredMeters* to *AllTokens* to represent the association between a meter and its unique token (*RegisteredTokensTypeInvariant*). • Addition of a new 'token' parameter and Guards to the *RegisterMeter* event to ensure the token is unique and has not previously been associated with a meter (*tokenUniquenessUnassignedGuard* and *tokenUniquenessAbsentGuard*). • Addition of a new 'token' parameter as well as Guards to *PayBill* and *AddMeterReading* requiring the correct associated token and meter parameters to be provided (*tokenAssignedToMeterGuard*).
[CR.6.] All meters should be checked as to whether they have fallen behind with submitting readings before the system's internal time may advance.	Guard added to *AdvanceTime* to not allow the system time to advance when Meters have failed to submit readings in the preceding 4 week period of time. This only applies once a Meter is out of its 12 week grace period (*noAdvanceHoldGuard*).
[CR.7.] Each bill must be unique.	Guard added to *GenerateBill* event to ensure that the bill parameter is not already known to the system (and therefore is unique) through checking if the bill parameter is a member of the *AllBills* variable (*billUniqueGuard*).

Table 5.6: Summary of integration steps for each critical requirement (continued)

Critical Requirement	**Adjustments made to model**
[CR.8.] All meters which are due to have bills generated shall have these generated in a timely manner.	Addition of Guard to *AdvanceTime* event to prevent system time from advancing while any meters do not have bills on a regular billing week (i.e. weeks 4, 8, 12, etc.) (*billsGeneratedHoldGuard*).
[CR.9.] Billing should be carried out with a fixed frequency.	A Guard has been added to the *GenerateBill* event which only allows billing to occur on weeks that are a multiple of 4 through use of the modulo operator (*CurrentWeekGuard*).
[CR.10.] Meters must specify which bill they are paying.	A Guard was added to supplement the existing Guards on *PayBill* events such that a bill must have been issued to a meter. This is done through domain-restricting the *MeterBill* variable (which maps *Meters* to their *Bills*) for a given meter parameter and then validating that the Bill appears within the range of the relation (*billAssignedToMeterGuard*).
[CR.11.] Only meters matching the disconnection criteria may be disconnected.	Existing constraints modelled (namely two Guards representing the two disconnection conditions which have been AND'd together) during the initial model meet this critical requirement. An additional Guard was added to the refined model as the author noticed that the *DisconnectMeter* event could be issued repeatedly - so meters may not be disconnected more than once (*meterNotAlreadyDisconnected*).
[CR.12.] A meter meeting the disconnection criteria must be disconnected in a timely manner.	A Guard has been expanded within the *AdvanceTime* event to prevent system time from advancing if any meters are eligible for disconnection (*noAdvanceHoldGuard*).
[CR.13.] Meters may only be disconnected if they are in registered state.	Existing Guard in *DisconnectMeter* event already ensured this critical requirement was met successfully (*meterTypeGuard*).

5.3.6.1 Challenges of representing critical requirements within the formal model

Representing the critical requirements within the formal model was a task for which many aspects of the Rodin toolset were utilised. This step involved the translation of critical requirements in natural language into the formalism required by the system model. Examples are given below, demonstrating challenges of integrating critical requirements, as well as the tool support that was used to resolve these challenges. A subsequent heading discusses the general approach to integrating critical requirements into the formal model.

DisconnectMeter occurs repeatedly This issue with *CR11 - Only meters matching the disconnection criteria may be disconnected* was revealed during ProB animation of the formal model. The existing guards within the *DisconnectMeter* event essentially met this critical requirement, however, during ProB animation, it was clear that the DisconnectMeter event was triggering multiple times for the same *meter* parameter. This was due to the fact that the event does not remove the meter from the *RegisteredMeters* variable, as a disconnected meter is still *registered* although it is marked as disconnected.

This resulted in any meters which met the disconnection criteria (as modelled in the *disconnectCriteria* guard) being subject to the DisconnectMeter event repeatedly. The use of ProB event traces enabled the author to notice this occurence and subsequently an additional guard was added (*meterNotAlreadyDisconnected*) which disables the event if the *meter* parameter has already been marked as disconnected.

AdvanceTime event improvements The AdvanceTime event was improved through integration of *CR6* and *CR8* which limited the situations in which the internal system time could advance, so as to ensure that bills were generated by the formal model on a regular schedule and that meters submitted readings in good time. This involved the implementation of two guards; the *billsGeneratedHoldGuard* and *noAdvanceHoldGuard* which represent predicates which only equate to true when meter readings have been submitted and billing has taken place (if it should have taken place during that *CurrentWeek*). The *noAdvanceHoldGuard* was then additionally expanded in scope by *CR12*, which included an additional restriction to prevent system time from advancing if meters were eligible for disconnection.

The construction of these predicates benefited significantly from the use of both animation and proof obligations.

Animation was utilised to ensure that time could not advance in circumstances where it should not (i.e. any meters had not submitted meters in over 4 weeks, that bills had

not been generated on schedule, and that meters eligible for disconnection remained connected). This was undertaken in two stages:

- Initially, the guards were formulated and the animation tool's option to choose which event was triggered next was utilised.
- When the guards appeared to be functioning correctly from this perspective (i.e. the *AdvanceTime* event was disabled at appropriate and correct points), a number of automatic *Random Animations* were used and the traces were manually examined to ensure that the *AdvanceTime* event was remaining disabled at appropriate points.

Once the guards appeared to be functionally correct, the **proving** view of the Rodin tool was used as each of these guards had an associated *well-definedness* proof obligation generated. This resulted in some minor additions to the guards in order to facilitate the discharging of these proof obligations.

General remarks on the use of the formal method in support of representing and refining critical requirements In the interest of not repeating the elements discussed in subsubsection 4.2.4.7, a summary of the primary representations that were utilised in integrating the case study critical requirements into its formal model is provided:

1. The primary representation of critical requirements was done through guards as many of the identified critical requirements related to the circumstances under which a control action may be issued and this is best represented as a guard due to guards being predicates which must all be true for the event to occur.
2. A smaller number of critical requirements were represented through invariants such as an invariant stating that the RegisteredMeters and RetiredMeters variables should have an intersection of the empty set. This was done to ensure that an event could not accidentally re-register a meter that had already been retired. Another collection of invariants were created related to the 'token' concept, which will be detailed below.
3. The integration of the 'token' concept required the creation of a new carrier set within the Context of the model, as well as a new variables, invariants and the adjustment of the existing collection of events to ensure that the notion of a token was fully integrated into various places within the model.

It is therefore clear, at least within the context of this case study, that the STPA focus on 'control actions' lends itself significantly to many critical requirements which will

translate cleanly into guards on the formal model and the critical requirements relating to elements of the process model tend towards being most-effectively represented by invariants. Larger scale critical requirements that propose entirely new components or aspects to the system design (as with the 'tokens' concept) may involve a combination of many formal specification elements together.

5.3.7 Step 7 - Causal factors analysis

To determine causal factors, the analysis begins by reviewing the existing functional control structure developed in subsection 5.3.2, in line with the guidance provided in subsubsection 4.2.4.8, as well as taking the existing set of hazards as identified in Table 5.5. Likely causal factor aspects[7] are then considered with regards to their ability to undermine the system.

One way of viewing this aspect of the analysis process is that many of the hazards specify some nebulous 'failure' which often at its root would be due to a miscommunication (or malicious subversion of communications) between components as described by the functional control structure or alternatively a given controller misunderstanding its controlled process or the state of another controller (which may once again be due to manipulation of the controller by a malicious actor). Performing causal factors analysis aims to contextualise these failures and provide further critical requirements in mitigation against them.

As an example to highlight this meaning, consider *H.23.* which states that *A meter is not disconnected when it should be disconnected.*. The critical requirement generated to address this hazard is as follows:

> *[CR.12.] A meter meeting the disconnection criteria must be disconnected in a timely manner.*

This critical requirement seeks to create a meaningful mitigation through requiring that system time may only advance once all meters eligible for disconnection have been disconnected, but this is only a mitigation of the immediate result of the hazard and does not deal with the circumstances of how this may occur.

Based on the functional control structure given in Figure 5.1, two scenarios are envisioned in which a meter may not be disconnected despite the controller deeming this the appropriate route to take and are presented below. Each of these scenarios are provided with exemplar critical requirements to demonstrate how these could be addressed.

[7] Though they are referred to as 'causal factors' within the table and throughout the text, these are also an additional form of hazard.

- The controller issues the disconnect command successfully but it is not received by the meter.
  - A critical requirement here would likely require acknowledgements to be sent by the meter to confirm it has actioned the disconnection of the electricity supply before the billing system ceases transmitting the disconnect command.
- The controller issues the disconnect command and it is received by the meter, which disregards it.
  - This hazard may be out-of-scope - there is not much that can be done by the billing system to ensure this occurs if the meter is disregarding commands due to either a malfunction or malicious tampering. However, the integration of anti-tamper systems into the meter may be an example of a suitable critical requirement to address this particular hazard.

These are both 'causal' factors in that they are contributing minor hazards towards a more significant hazard. Causal factors may arise which are essentially unmitigable, however, the identification of them is important for assurance and traceability purposes as they may highlight some innate issue with the system design or architecture that may require reconsideration.

Further example causal factors relating to the system under analysis, as well as their associated hazards, are detailed in Table 5.7 as well as what critical requirements may be generated in order to address them.

Table 5.7: Causal factors and potential critical requirements

Causal factor	**Related hazards**	**Sample critical requirements**
Control actions are issued but not successfully received - i.e. the communication channel between the Meter and BillingSystem entities is not reliable at the transport layer.	H.11., H.15., H.16., H.23.	• Requirement that all commands between the Meters and BillingSystem entities have a confirmation sent after being received in order to ensure that all commands are successfully received. • Stipulation of a reliable transport layer between the entities.

Table 5.7: Causal factors and potential critical requirements (continued)

Causal factor	Related hazards	Sample critical requirements
Tokens become known to third-parties allowing for malicious or fraudulent actions to be undertaken on behalf of meters.	H.14. and H.20.	• Requirement that Meters / BillingSystem utilise asymmetric encryption or similar, in addition to; • A key management infrastructure to ensure that keys can be regularly rotated and allow revocation of compromised keys.
Compromise of the Meter's measuring sensors such that readings out of line with reality may be transmitted.	Related to H.13.	• Anti-tampering systems on the Meter to ensure that any attempts to tamper with the meter causes it to either attempt to notify the BillingSystem or cease working.

There are several categories of hazards which, while not mitigable on the case study without extensive decomposition and significantly more detail in the initial case study, have not been considered.

5.3.8 Step 8 - Iteration and scoping

Given the scale of the provided system, iteration/scoping into sub-systems was not undertaken due to the lack of low-level detail available in the synthetic case study. Were such an analysis to be undertaken, it would likely begin as detailed in Figure 5.2 where each box would encompass a sub-system of the overall whole. The analysis would then be carried out once in its entirety per sub-system (with each sub-system's formal model being composed into a single model for the purpose of validating overall system behaviour is as defined in the system documentation).

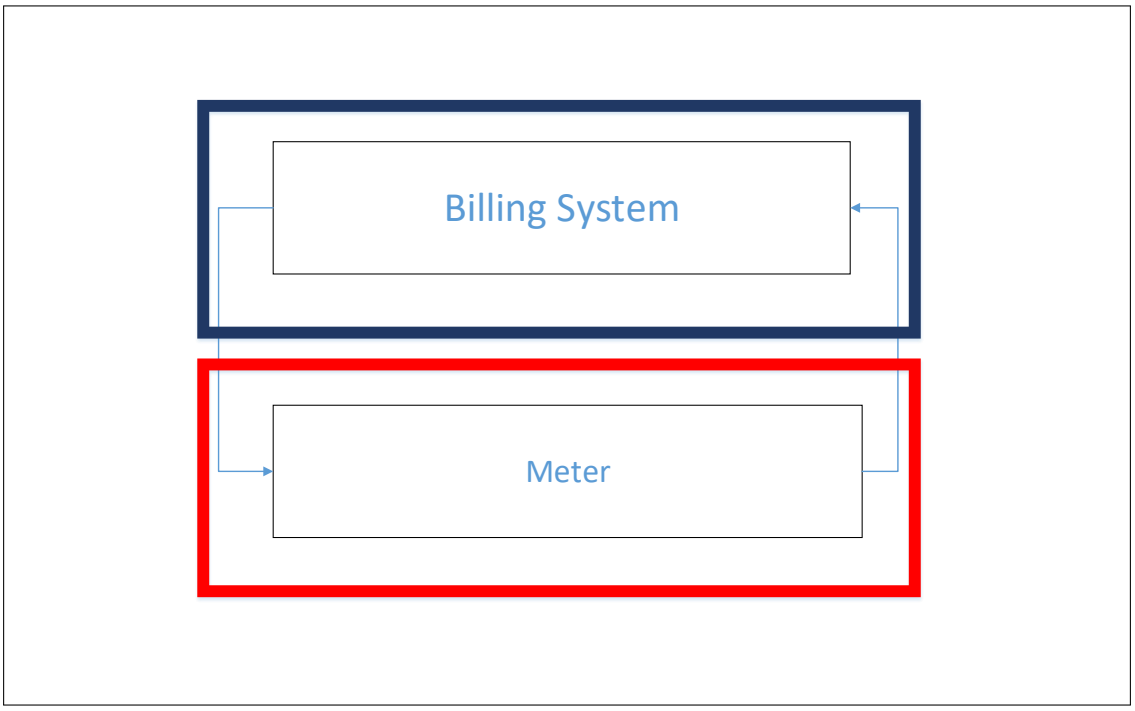

Figure 5.2: Scoping areas/identification

5.4 Lessons learned from the smart meter case study

The smart meter case study presented an opportunity to gain an understanding of the positive and negative aspects of the early version of SE-STPA. These 'lessons learned' were then used to inform the development of the methodology into the version presented in Chapter 4. The lessons learned, as well as the modifications made to SE-STPA, are given in this section.

5.4.1 Security analysis issues and improvements

The second version of SE-STPA, as applied to this smart meter case study, concentrated security analysis into two steps of the analysis:

- **Step 5** - Hazard analysis and critical requirement generation.
- **Step 7** - Causal factors analysis

While each of these steps allowed for the identification of security issues relating to control actions, it did not permit analysis of security issues more broadly (i.e. where these security issues did not necessarily impinge on the control actions themselves). For instance, one can consider the CIA triad [116] as an example of possible classifications of security issues:

- **Confidentiality**, the practice of restricting information which does not need to be widely known, may have no impact on the safety-related functioning of the system, but may be critical to maintaining as small an attack surface as possible from a security perspective.

  – In the case of the smart meter case study, sensitive information such as billing information may be being exchanged over communications channels. Equally, there may be an aspect of proprietary or otherwise commercially-sensitive information being exchanged between entities of the system. This is not a feature of control action analysis or causal factors analysis as applied to this case study.

  The exchange of this information unencrypted, 'in the clear', represents a possible confidentiality concern but does not make itself known during either the hazard analysis or causal factors analysis as it is unlikely to result in a control action becoming hazardous. This means that **confidentiality is not adequately considered by this version of the methodology**.

- **Integrity**, the practice of ensuring that data is correct, trustworthy and attributable to known sources[8], represents a crossover of both security and safety concerns.

  – Within the existing case study, *CR5* is an example of a critical requirement generated in response to a number of hazards such as *H14* and *H20*. These hazards relate to control actions being incorrectly processed as being attributable to another smart meter than the one that is actually intended. This is an issue from both a security and safety perceptive, as disconnection of an incorrect smart meter is an unsafe action, and if done maliciously, could form part of a broader attack on an individual.

  This aspect is partially covered by the existing methodology, but there are aspects that are not currently considered. For instance, the *integrity* of the smart meters is not considered. These are smart devices that reside within individual's homes, and possess networking and other capabilities that could make them ideally positioned for cyber attacks. If a smart meter device was utilised to spread malware to the user's home network, and also continued to behave normally within the context of being a smart meter from the perspective of the Billing System, this would not be captured by the current methodology. Other threats, such as smart meters being used for crypto-currency mining, are also outside of the scope of the existing analysis. There are therefore **aspects of integrity which are not currently considered by the methodology as applied in this case study**.

[8]Non-repudiation is included in this category for brevity, although other interpretations of the CIA Triad have it as a separate category.

- **Availability**, the practice of ensuring that the system functions are available to end users and other stakeholders, is primarily a safety concern for cyber-physical systems.
  - Within the existing case study, *CR11* was generated to address *H22* which is an example of a critical requirement generated to ensure that parts of the system that should remain active and available are maintained in this state.

  This aspect is well considered by the existing control action analysis and causal factors analysis steps, as control actions are the primary means through which availability manifests. **Availability is therefore well considered by the methodology as utilised in this case study**.

In response to this 'lessons learned' analysis, the **adversarial modelling** step was added to SE-STPA. The justification for the development and inclusion of this step into SE-STPA is justified in subsection 3.7.2 and general detail on the exact process of applying the new step can be found in subsubsection 4.2.4.6. The next section considers application of adversarial modelling to the case study retrospectively, in order to demonstrate the advantage this step provides to the methodology in its consideration of security issues.

5.4.1.1 Adversarial modelling as applied to the smart meter case study

Adversarial modelling as applied to the case study involves first the annotation of the manipulation points within the functional control structure, followed by identification and specification of possible adversaries, and then generation of critical requirements as required to mitigate the adversarial behaviours. The annotation of the functional control structure can be found in Figure 5.3, which is a restatement of Figure 4.3.

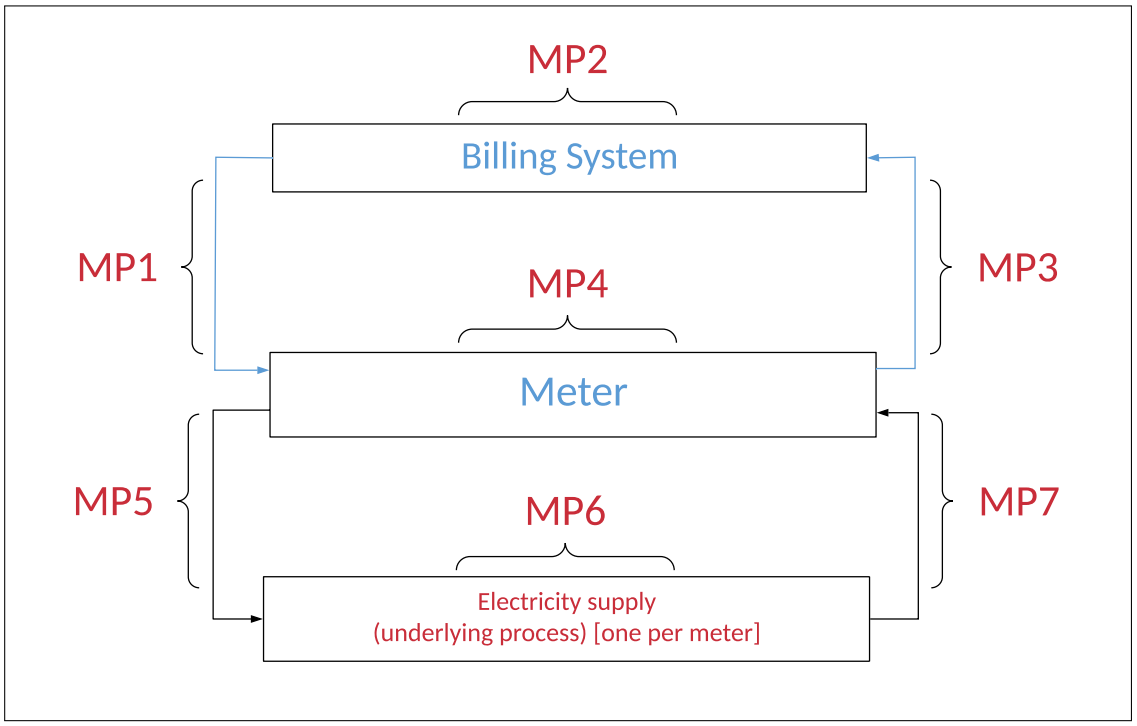

Figure 5.3: Annotated functional control structure for smart meter, with manipulation points.

From here, we can identify two major adversaries who are reasonable when one considers the function of the system:

1. **Fraudulent consumers**, who wish to pay less on their electricity bill.
2. An **insider threat** with access to the Billing System, who has been unduly influenced to disrupt the activities of the smart meter infrastructure.

Fraudulent consumer This adversary is a capture of a fairly simplistic threat to the purpose of the smart meter system from a security perspective. The main threat actor involved is a technically-knowledgeable consumer who has access to a physical smart meter on their premises, and so is able to effectively 'reduce' their electricity bill through tampering with the meter's interface to the electricity supply. The adversary is further detailed in Table 5.8.

Table 5.8: *Fraudulent consumer* adversary description

Property	*Detail*
Identifier	Adversary 1 (A1)
Name/categorisation	Fraudulent consumer

Table 5.8: *Fraudulent consumer* adversary description (continued)

Property	*Detail*
Intent	Adversary intends to reduce their bill through tampering with the meter.
Access	Adversary can see and manipulate MP5 and MP7.
Information	Adversary is aware that feedback channel MP7 reports electricity usage by the consumer.
Actions	Single step: Adversary intercepts and modifies MP7 result at all times via some sort of interception device. This has the net effect of putting the electricity consumption, as recorded by the smart meter, below what has actually been used.

The generation of critical requirements to address *Adversary 1* can be found in Table 5.9.

Table 5.9: Critical requirement generation to address *Fraudulent Consumer* adversary

Hazard	*Generated critical requirement*
H26: Adversary manipulates data between electricity supply and meter to reduce reported electricity usage.	**CR14:** Meter will receive local average usage and will raise an alert if readings are more than 25% below this in a month's period. **CR15:** Billing System shall flag any meters that send alerts or provide readings that are more than 20% below the projections for that meter over a month's period. **CR16:** Meter's interface to the electricity supply shall involve tamper-evident seals which can be manually inspected when meter readings are suspected.

Insider threat The insider threat adversary represents an individual who works in an administrative capacity with the Billing System, who has been influenced (either through blackmail, bribery or other mechanism) to interfere with the correct functioning of the broader smart meter architecture. This adversary therefore has extensive access to the system, and has been manipulated into modifying the Billing System such that all messages to Meters shall include malware which will then be installed on the Meters.

Table 5.10: *Insider threat* adversary description

Property	*Detail*
Identifier	Adversary 2 (A2)
Name/categorisation	Insider threat
Intent	Adversary intends to spread malware through smart meter architecture having been influenced by some external threat actor.
Access	Adversary has administrative access to MP2 and can manipulate it extensively. This includes modification of MP1, which is constructed automatically by MP2 based on a proprietary protocol.
Information	Adversary has full understanding and visibility of all information flowing across MP1, MP2 and MP3.
Actions	Single step: Adversary modifies MP2 such that malware is included in all communications from the Billing System to Meters.

The generation of critical requirements to address *Adversary 2* can be found in Table 5.9.

Table 5.11: Critical requirement generation to address *Insider Threat* adversary

Hazard	*Generated critical requirement*
H27: Adversary embeds malware into all communications from Billing System to Meters.	**CR17:** Modifications to MP2 from authorised staff shall operate on a 'two-man rule' such that one staff member proposes a modification, and another must confirm it. **CR18:** A log shall be kept, held at a higher privilege level than system administrators, to track all modifications to the Billing System made by system administrators.

5.4.1.2 Analysis

The benefit of the adversary modelling step is that it enables the analyst to consider 'pure' security threats that do not necessarily affect the proper functioning of the smart meter system. This is demonstrated through the analysis of *Adversary 2* who is negatively influencing the system in terms of security, but this may have no knock-on effect from a safety perspective.

Undertaking this within the existing methodology framework (i.e. the second version of SE-STPA) would involve one of the following:

1. Having to contemplate an extensive list of all possible adversary types and actions while undertaking control action analysis (Step 4), such that all control actions are considered in all situations where security of the system may have been degraded. This would introduce a significant burden on this step, and would complicate the control action analysis. The current approach of considering how control actions may be hazardous where 'hazardous' can mean both *unsafe* and *insecure* enables thoughtfulness around individual control actions and how security/safety issues can arise through the control actions, without being too unwieldy to apply.

    - The insider threat adversary (A2) proves a good example for testing this approach; to consider it in the control action analysis context, one would need to consider the hazard of *'control action contains malware'* as part of the *Is issued* context. This hazard could apply to every control action in every system, and therefore consideration of it, as well as all other security hazards/scenarios, as part of the control action analysis stage is onerous.

2. Placing this degree of security analysis within the causal factors analysis would once again focus on *control actions* and while it may be possible to consider adversaries or degradation in the security of the system within this notion, this is not the intent of this step. Causal factors analysis is intended to consider what possible influencing factors exist and contribute to a control action being issued in a situation that made it unsafe or insecure (i.e. hazardous). This means that the causal factors analysis does not ordinarily consider situations where control actions are *not* hazardous, but the system security is degraded nonetheless.

    - The insider threat adversary (A2) also demonstrates why this approach is not efficient or useful. Causal factors is usually undertaken with a standard 'diagram' that models a large number of the contributory factors that can result in control actions becoming hazardous, based on analyst experience and guidance from the literature. To include all possible adversary actions or influences within this step (which is essentially unbounded) would not scale efficiently when compared to the adversarial modelling approach.

The difference therefore is in fundamental approach; the notion of including this level of security/threat modelling in either of the two existing steps would involve consideration of an unbounded number of possible adversaries and their actions in a *reactive* sense, while adversarial modelling focuses the analysis on *proactive* identification of a number of adversaries and their most likely attack paths through the system to achieve their goals. This bounds the problem and generates an artefact which aids traceability when writing future security documentation.

5.4.2 Sequencing of methodology steps

The sequencing of steps within the second version of SE-STPA also represented a shortcoming during the smart meter case study. The concentration of security analysis within the causal factors stage meant that a large number of *security* critical requirements would be generated once the analysis was in its last stages and therefore could not be validated through the formal model.

In order to resolve this, adversarial modelling (as the 'major' step for consideration of security issues) was placed after control action analysis and prior to the integration of critical requirements into the formal model. This ensured that the bulk of critical requirements were generated prior to the critical requirements being integrated into the formal model. This can be seen in how the steps are detailed in subsection 4.2.4.

Chapter 6

UAV case study

6.1 Introduction & outline of case study

For the second case study, a scaled down version of existing work on a multi-UAV (unmanned aerial vehicle) system [19] was considered. This system manages a number of aircraft, with a ground control station serving as the interface between the aircraft and human operators who seek to task aircraft with activities and routes. The ground control station may also be used by human operators to plan routes and validate whether these are valid and within the constraints of the system.

The system therefore consists of three entities that can be immediately identified:

- *Operators* which are human actors within the system who provide either plans or commands to the Ground Control Station which will be validated in the case of the former and the result of validation fed-back, or acted upon in the case of the latter (so long as they pass validation).
- The *Ground Control Station* which contains an interface to both the human operators and to the aircraft, as well as an intelligent planning system which receives input from operators and attempts to plan routes for the aircraft to satisfy this input (within the set of constraints that exist around restricted airspace and ensuring minimum safe separation distances).
- A number of *Aircraft* which attempt to fulfill their routes with deviations allowed for obstacle avoidance.

Several assumptions can also be made, as detailed below:

1. There is always at least **one** aircraft attached to the system.

2. There is always at least **one** operator interacting with the system.
3. The ground control station always exists.
4. There is **no more than one** ground control station at any one time.

This level of detail will suffice in allowing the first pass of analysis using SE-STPA to be undertaken.

6.2 Application of SE-STPA

6.2.1 Step 1 - Establishing the system engineering basis

The basis for performing the analysis has been reasonably scoped already by the identification of entities and the assumptions made when outlining the case study in Section 6.1. From this information, it is possible to then derive a system purpose statement, as follows:

> The system shall maintain adequate and necessary separation between obstacles and aircraft, as well as between aircraft. In addition to this, the system shall maintain complete control of UAVs at all stages of flight, as well as avoiding no-fly zones and fulfil all other requirements of the Civil Aviation Authority, with particular reference to the Air Navigation Order 2016 (ANO 2016) [30].

It is then possible to arrive at a set of system losses, derived from failure to meet any aspect of the system purpose:

L1: Separation between aircraft and environment falls below threshold, causing damage to aircraft and/or environment.

L2: Separation between aircraft is not maintained, damaging aircraft.

L3: Aircraft deviate from flight plan without notifying GCS.

L4: GCS is uncertain of current state of any aircraft at any point.

L5: Aircraft violate any restrictions as defined in the Air Navigation Order 2016 while in transit.

The generation of these losses permits the further derivation of top-level (system) hazards to be done:

H1: Aircraft fail to detect and route around other aircraft or the environment.

H2: Aircraft deviate to compensate for a change in flight conditions without notification to the GCS.

H3: Aircraft do not report position and other telemetry data to GCS sufficiently frequently while in any stage of flight.

H4: Aircraft enter restricted airspace.

H5: GCS provides routes to aircraft that violate regulations if flown.

H6: Operators provide routes which cannot be fulfilled without violation of regulations.

The goal here is to identify sufficiently-broad losses and hazards which work at a system-level to guide future analysis, and allow the relation of more localised or specialised hazards to be made to system-level hazards and their associated losses.

The final aspect of this step involves the multi-dimensional mapping of losses and hazards which can be found in Table 6.1.

Table 6.1: Hazards and losses mappings for UAV case study

Loss	**Hazard**
L1: Separation between aircraft and environment falls below threshold, causing damage to aircraft and/or environment.	**H1:** Aircraft fail to detect and route around other aircraft or the environment. **H2:** Aircraft deviate to compensate for a change in flight conditions without notification to the GCS. **H3:** Aircraft do not report position and other telemetry data to GCS sufficiently frequently while in any stage of flight.
L2: Separation between aircraft is not maintained, damaging aircraft.	*See **L1** for hazards associated with this loss.*
L3: Aircraft deviate from flight plan without notifying GCS.	**H2:** Aircraft deviate to compensate for a change in flight conditions without notification to the GCS. **H3:** Aircraft do not report position and other telemetry data to GCS sufficiently frequently while in any stage of flight.

Table 6.1: Hazards and losses mappings for UAV case study (continued)

Loss	Hazard
L4: GCS is uncertain of current state of any aircraft at any point.	*See **L3** for hazards associated with this loss.*
L5: Aircraft violate any restrictions as defined in the Air Navigation Order 2016 while in transit.	***All** identified hazards apply to this loss.*

6.2.2 Step 2 - Building the control structure

At this stage, it is necessary to build a control structure. This permits the analysis to consider the interaction between the three entities already identified in Section 6.1, with the intent of identifying specific control actions which are needed in latter steps of the analysis.

This begins with an overall functional control structure diagram which maps the three entities and the interactions between them, which can be found in Figure 6.1. This will then enable the next step of analysis.

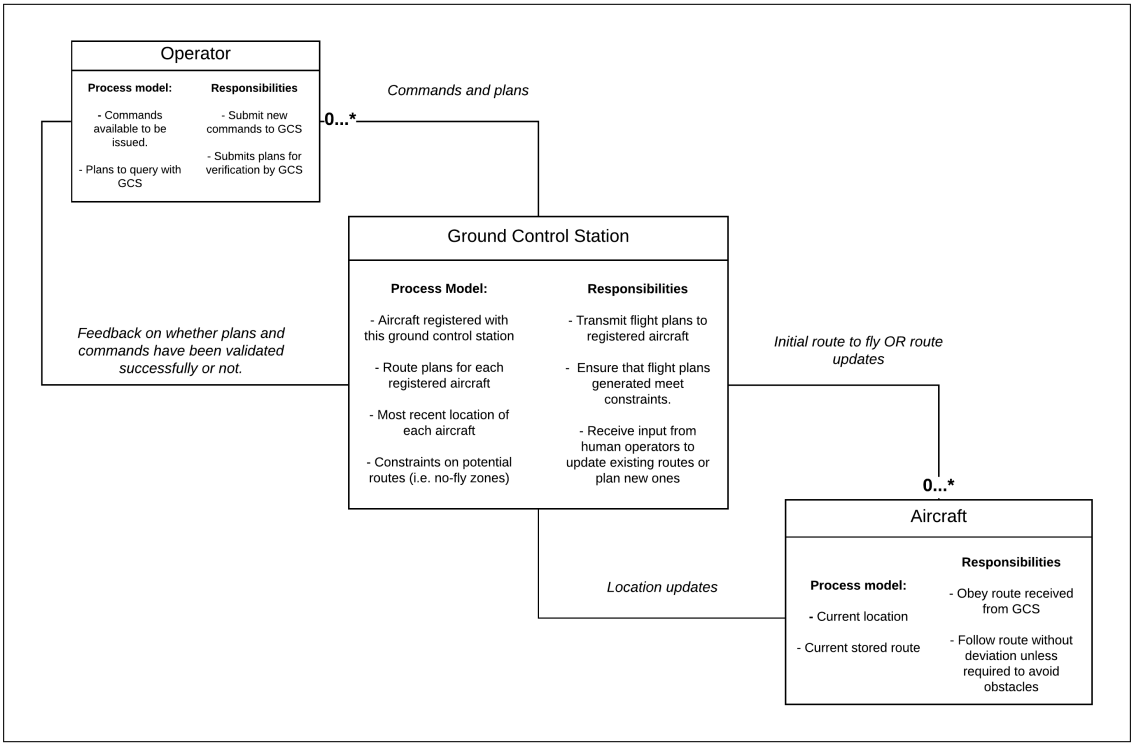

Figure 6.1: Overall functional control structure for UAV case study.

6.2.3 Step 3 - Identify control actions

The identification of control actions is enabled by scoping further into the functional control structure and considering each 'pair' of entities. This is possible due to the fact there is only communication between operators and aircraft via the ground control system, allowing the interactions on either side of the GCS to be isolated and considered.

The visual representation of this can be found in Figure 6.2 and Figure 6.2.

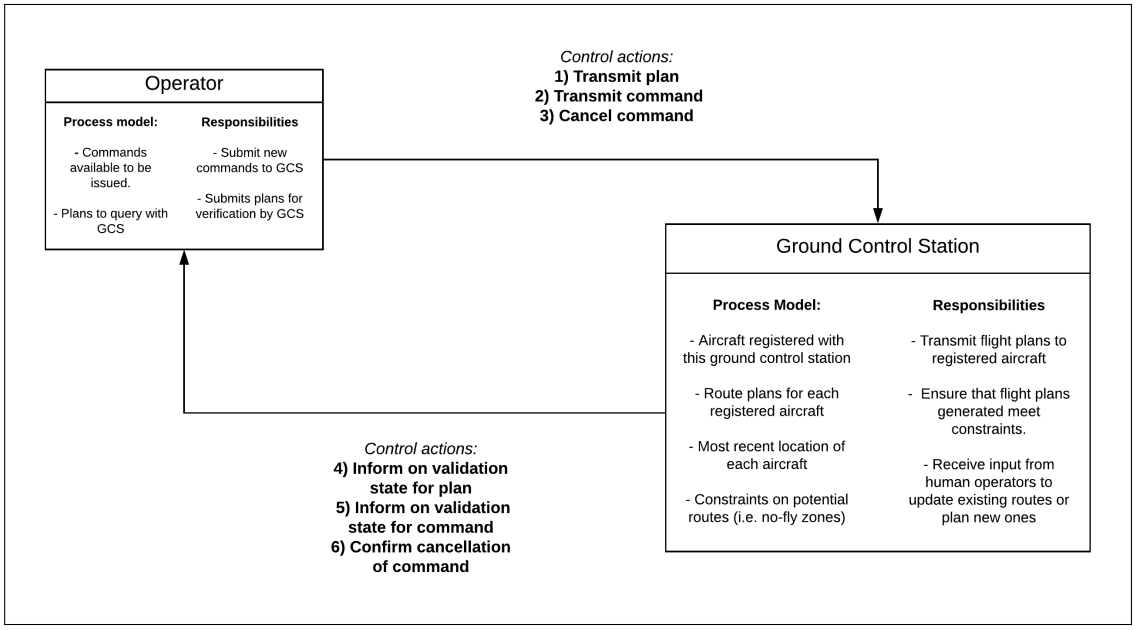

Figure 6.2: Scoped functional control structure between operators and GCS.

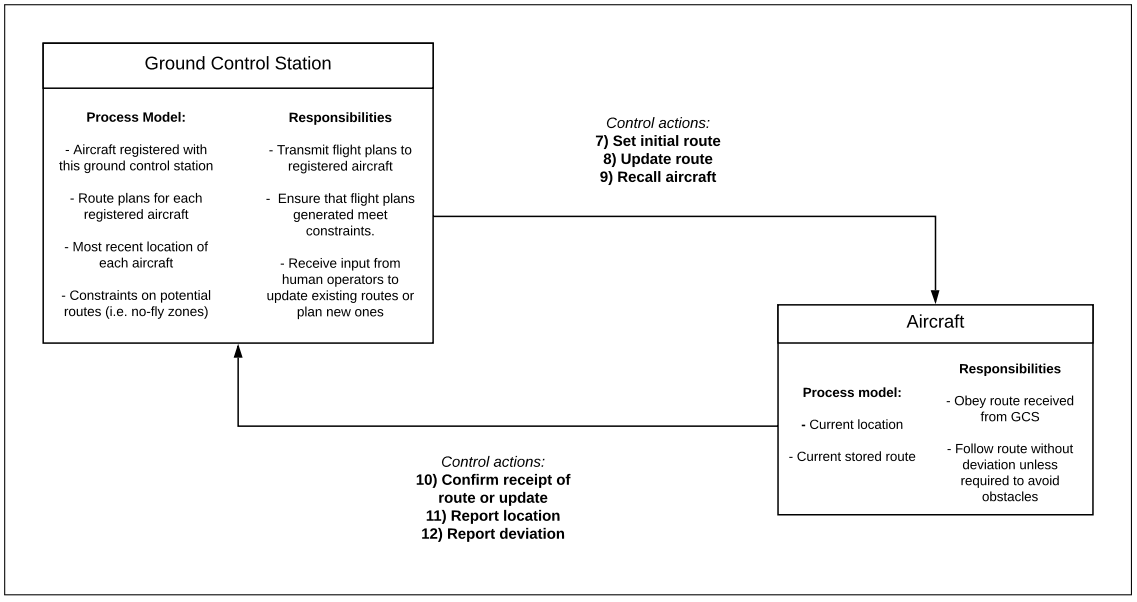

Figure 6.3: Scoped functional control structure between GCS and aircraft.

The control actions extracted from this visual process, and their broad descriptions, can be found in Table 6.2.

Table 6.2: Control actions extracted from functional control structure for UAV case study

#	Control action	Description
1	*Transmit plan*	The operator provides a plan to the GCS to be validated and awaits the result of this validation.
2	*Transmit command*	The operator provides a plan to the GCS with the expectation it is enacted immediately, and awaits the result of validation.
3	*Cancel command*	The operator requests cancellation of an existing command via the GCS.
4	*Report validation state of existing plan*	The GCS runs the validation process on a plan provided previously by an operator and reports whether the plan validates successfully (i.e. there are no violations or restricted aspects to the plan).
5	*Report validation state of existing command*	The GCS runs the validation process on the provided plan. If validation passes successfully, the GCS will report validation status to the operator.
6	*Confirm cancellation of a command*	The GCS confirms to the operator requesting a cancellation whether the cancellation was successful, or whether it has failed (i.e. as the command was already completed).
7	*Set initial route*	The GCS provides an aircraft with a validated command/plan and instructs the aircraft to begin the route immediately.
8	*Update route*	The GCS sends an adjustment in some way to the existing route an aircraft possesses and is currently following.

Table 6.2: Control actions extracted from functional control structure for UAV case study (continued)

#	Control action	Description
9	*Recall aircraft*	The GCS instructs an aircraft to return to its starting point immediately and disregard any existing route it was following.
10	*Confirm receipt of route or update*	The aircraft notifies the GCS that it has fully received and processed either the initial route it has been provided with or has processed an in-flight update provided by the GCS into its future route.
11	*Report location*	The aircraft provides a periodic update of its location to the GCS such that the GCS can be assured of an aircraft's progress within its planned route.
12	*Report deviation*	The aircraft informs the GCS whenever it makes an unplanned diversion or substantial change to the flight path due to external factors (i.e. unexpected obstacle, poor weather).

This identification once again facilitates the next step of analysis.

6.2.4 Step 4 - Construction of initial formal model

An overview is provided in this section of the formal model used as part of this case study. It will provide a general outline of the major variables and events and how these relate to the preceding steps of the analysis. Based off the results from Step 3, the initial formal model consisted of the following major events:

1. The **INITIALISATION** event, providing default instantiations for all variables used by the model.
2. The **RegisterAircraft** event, which enables a parameter from the carrier set of *DRONES* - which has not already been registered - to be added to the fleet of registered aircraft (as represented by the *RegisteredAircraft* variable).

3. The **Transmit** events (*TransmitPlan* and *TransmitCommand*) which model operators lodging plans and commands with the Ground Control Station. Plans and commands must first be lodged with the Ground Control Station before they can be validated.

4. The **Validate** events (*ValidatePlan* and *ValidateCommand*) which serve to change the validation state of plans/commands which are valid (i.e. they do not contain any restricted locations) to formally being *valid*.

5. A number of events which model the **tasking of aircraft** (*TaskAircraft*, *RetaskAircraft*, *RecallAircraft* and *CancelCommand*) by setting or modifying an aircraft's route. Aircraft routes are tracked via the *AircraftRoutes* variable.

6. The **Report** events (*ReportLocation* and *ReportDeviation*) which represent the aircraft reporting back its current location and/or any deviations which have to occur to its route, such that the Ground Control Station has a record of all locations that an aircraft visits.

7. The **Confirm** events (*ConfirmReceipt* and *ConfirmCancellation*) which represent the feedback from an aircraft in response to being tasked by the GCS.

8. The **Query** events (*QueryPlanValidationState* and *QueryCommandValidationState*) which model an operator requesting the validation status of a command or plan.

The major variables in use in the model are as follows:

1. **Plans** and **Commands** variables which represent plans and commands held by the GCS; typed as subsets of the powerset of the *LOCATIONS* carrier set that is defined in DroneContextInitial.

2. **RegisteredAircraft** variable which models all aircraft registered with the GCS; typed as a subset of the *DRONES* carrier set present in the Context.

3. **RestrictedLocations** variable which models locations that should not be present in aircraft routes; typed as a subset of the *LOCATIONS* carrier set.

4. **PlanValidationState** and **CommandValidationState** which represent the status of plans and commands and are mapped to a validation state. This is achieved in the model as a partial function, mapping the respective set (e.g. Commands) to *VALIDATIONSTATE*, which is a carrier set defined in DroneContextInitial. This carrier set is partitioned into three constants via an axiom:

    - VALID.
    - INVALID.

- UNDETERMINED.

5. **AircraftRoutes** variable, which maps registered aircraft to a command; typed as a partial function between the *RegisteredAircraft* variable and the *Commands* variable.

The complete formal model for this step can be found in Appendix C. This model will be refined and improved through later steps.

6.2.5 Step 5 - Control action analysis & critical requirement generation

6.2.5.1 Control action analysis

The control actions generated in subsection 6.2.3 and modelled in subsection 6.2.4 are then subject to analysis in order to identify any hazards, using a slightly modified form of the STPA analysis table as detailed in subsubsection 4.2.4.5. The fourth column - *Control action is issued for incorrect duration* - is included though it can be omitted in the circumstance in which all control actions are discrete and therefore have no notion of *duration*.

The results of this control action analysis are presented in Table 6.3.

Table 6.3: UAV control action analysis results

Control Action	*Is issued*	*Is not issued*	*Is issued out of sequence*	*Is issued for incorrect duration*
Transmit Plan	*No hazards found.*	*No hazards found.*	Transmitting a plan while another plan is already undergoing validation may leave the system in an indeterminable state. **[H7]**	N/A - transmitting plan is a discrete action.

Table 6.3: UAV control action analysis results (continued)

Control Action	*Is issued*	*Is not issued*	*Is issued out of sequence*	*Is issued for incorrect duration*
Transmit Command	Command containing unsafe waypoints is transmitted. [**H8**]	Aircraft receive no commands. [**H9**]	Transmitting command while another command is undergoing validation may leave the system in an indeterminable state. [**H10**]	N/A - transmitting command is a discrete action.
Cancel Command	Cancelling a command before any command is issued. [**H11**]	*No hazards found.*	A command is cancelled after an aircraft has already completed it. [**H12**]	N/A - cancelling a command is a discrete action.
Report Validation State Of Plan	Plan validation state is reported incorrectly. [**H13**]	Plan validation state being unreported causes operator confusion. [**H14**]	Plan validation state reported prior to validation being complete. [**H15**]	N/A - reporting validation state is discrete.
Report Validation State of Command	Command validation state is reported incorrectly. [**H16**]	Command validation state being unreported causes operator confusion. [**H17**]	Command validation state reported prior to validation being complete. [**H18**].	N/A - reporting validation state is discrete.

Table 6.3: UAV control action analysis results (continued)

Control Action	*Is issued*	*Is not issued*	*Is issued out of sequence*	*Is issued for incorrect duration*
Confirm Cancellation of a Command	Command cancellation confirmation is issued before aircraft ceases following route. [**H19**]	Cancellation is never confirmed. [**H20**]	Confirmation of cancellation issued when cancellation has not been requested. [**H21**]	N/A - confirmation of cancellation is discrete.
Set Initial Route	Initial route is set containing restricted or hazardous waypoints. [**H22**]	Commands are not executed if routes are not set. [**H23**]	Initial route is set before command passes validation. [**H24**]	N/A - setting route is a discrete action.
Update Route	Updated waypoints contains restricted or hazardous waypoints. [**H25**]	Existing route becomes restricted/hazardous in some way and is not corrected by an update. [**H26**]	Update is sent before initial route is set [**H27**]	N/A - update route is a discrete action.
Recall Aircraft	Aircraft recalled when recall would cause aircraft to enter restricted/hazardous airspace. [**H28**]	Aircraft not recalled in response to existing route becoming hazardous/restricted. [**H29**]	Aircraft recalled while not on a route. [**H30**]	N/A - recalling aircraft is a discrete action.

Table 6.3: UAV control action analysis results (continued)

Control Action	*Is issued*	*Is not issued*	*Is issued out of sequence*	*Is issued for incorrect duration*
Confirm receipt of route or update	Aircraft confirms receipt of update/route when GCS has not issued an update/route to aircraft. **[H31]**	Aircraft fails to confirm receipt of update/route. **[H32]**	***[H31]*** *encapsulates hazards in this category also.*	N/A - confirm receipt is discrete.
Report location	Aircraft incorrectly reports location. **[H33]**	Aircraft fails to report location. **[H34]**	Aircraft reports locations in an irregular sequence. **[H35]**	N/A - reporting location is discrete.
Report deviation	Aircraft incorrectly reports deviation. **[H36]**	Aircraft fails to report deviations. **[H37]**	Aircraft reports deviations in an irregular sequence. **[H38]**	N/A - reporting deviations is discrete.

6.2.5.2 Critical requirement generation

At this stage, the generated hazards are then 'sifted' to see if they fall within the scope of the analysis as some may not. The ones that are deemed to be in scope will then have a critical requirement generated for them, which will be taken forward in the process.

The results of this process can be found in Table 6.4.

Table 6.4: UAV hazards & resultant critical requirements

Hazard & identifier	*In scope?*	*Resulting critical requirement/justification*
Transmitting a plan while another plan is already undergoing validation may leave the system in an indeterminable state. [**H7**]	In-scope	Transmitted plans will be entered into a validation queue and shall be validated in order of submission. [**CR1**]
Command containing unsafe waypoints is transmitted. [**H8**]	In-scope	Commands must not contain unsafe waypoints. [**CR2**]
Aircraft receive no commands. [**H9**]	Out-of-scope in general terms	Aircraft may receive no commands when there are simply no commands to be given during a specified time-frame; this is therefore not within scope as this is a *business* decision[1].
Transmitting command while another command is undergoing validation may leave the system in an indeterminable state. [**H10**]	In-scope	Transmitted commands will be entered into a validation queue and shall be validated in order of submission. [**CR3**]
Cancelling a command before any command is issued. [**H11**]	In-scope	Cancellation of a command must specify a command which already exists. [**CR4**]
A command is cancelled after an aircraft has already completed it. [**H12**]	In-scope	Cancellation may only occur for commands which are not already complete. [**CR5**].
Plan validation state is reported incorrectly. [**H13**]	In-scope	Plan validation states must always be reported accurately. [**CR6**]
Plan validation state being unreported causes operator confusion. [**H14**]	In-scope	Plan validation state must always be reported when requested. [**CR7**]
Plan validation state reported prior to validation being complete. [**H15**]	In-scope	Plan validation state must always be reported as *undetermined* if requested prior to validation being complete. [**CR8**]

[1]This hazard may be revisited during causal factors analysis (subsection 6.2.8) however

Table 6.4: UAV hazards & resultant critical requirements (continued)

Hazard & identifier	*In scope?*	*Resulting critical requirement/justification*
Command validation state is reported incorrectly. [**H16**]	In-scope	Command validation state must always be reported accurately. [**CR9**]
Command validation state being unreported causes operator confusion. [**H17**]	In-scope	Command validation state must always be reported when requested. [**C10**]
Command validation state reported prior to validation being complete. [**H18**]	In-scope	Command validation state must always be reported as *undetermined* if requested prior to validation being complete. [**CR11**]
Command cancellation confirmation is issued before aircraft ceases following route. [**H19**]	In-scope	Command cancellation confirmation is not to be issued until aircraft has ceased following route. [**CR12**]
Cancellation is never confirmed. [**H20**]	In-scope	Command cancellation state must always be reported. [**CR13**]
Confirmation of cancellation issued when cancellation has not been requested. [**H21**]	In-scope	Command cancellation confirmation must only ever be issued in response to a cancellation request. [**CR14**]
Initial route is set containing restricted or hazardous waypoints. [**H22**]	In-scope	Initial route must not contain restricted or hazardous waypoints. [**CR15**]
Commands are not executed if routes are not set. [**H23**]	In-scope	Commands passing validation must be passed to aircraft in the form of routes. [**CR16**]
Initial route is set before command passes validation. [**H24**]	In-scope	Aircraft may not have their initial routes set to commands that do not pass validation. [**CR17**]
Updated waypoints contains restricted or hazardous waypoints. [**H25**]	In-scope	Updates provided to aircraft may not contain restricted or hazardous waypoints. [**CR18**]
Existing route becomes restricted/hazardous in some way and is not corrected by an update. [**H26**]	In-scope	Routes that aircraft are currently on should be updated to remove hazardous waypoints or return aircraft home. [**CR19**]

Table 6.4: UAV hazards & resultant critical requirements (continued)

Hazard & identifier	*In scope?*	*Resulting critical requirement/justification*
Update is sent before initial route is set. [**H27**]	In-scope	Aircraft may not be updated until provided with an initial route. [**CR20**]
Aircraft recalled when recall would cause aircraft to enter restricted/hazardous airspace. [**H28**]	In-scope	Aircraft may not be recalled when to do so is hazardous [**CR21**]
Aircraft not recalled in response to existing route becoming hazardous/restricted. [**H29**]	In-scope	*Addressed by **[CR19]***
Aircraft recalled while not on a route. [**H30**]	In-scope	Aircraft may only be recalled while on a route. [**CR22**]
Aircraft confirms receipt of update/route when GCS has not issued an update/route to aircraft. [**H31**]	In-scope	Aircraft may only confirm receipt of update/initial route when provided one by the GCS. [**CR23**]
Aircraft fails to confirm receipt of update/route. [**H32**]	In-scope	Aircraft will always confirm receipt of update/initial route when one is provided by the GCS. [**CR24**]
Aircraft incorrectly reports location. [**H33**]	In-scope	Aircraft must report accurate, regular location data. [**CR25**]
Aircraft fails to report location. [**H34**]	In-scope	*Addressed by **[CR25]***
Aircraft reports locations in an irregular sequence. [**H35**]	In-scope	Aircraft must report locations with a time-stamp to prevent sequencing issues. [**CR26**]
Aircraft incorrectly reports deviation. [**H36**]	In-scope	Aircraft must report accurate deviations where these occur. [**CR27**]
Aircraft fails to report deviations. [**H37**]	In-scope	*Addressed by **[CR27]**.*
Aircraft reports deviations in an irregular sequence. [**H38**]	In-scope	Aircraft must report deviations with a time-stamp to prevent sequencing issues. [**CR28**]

6.2.6 Step 6 - Adversary modelling & critical requirement generation

This step of the analysis focuses on the identification of *likely* adversaries that the system may encounter during its operation. Overt identification of such adversaries and consideration of their potential actions enables the secure-by-design concept to be realised. The adversary profile is leveraged to generate critical requirements through likely attack or manipulation paths that are identified as part of this step.

6.2.6.1 Annotating the control structure

The first aspect of this step involves the annotation of the existing functional control structure for the system with 'manipulation points' so as to enable the consideration of manipulation/attack paths through the system. A fully annotated version of the functional control structure can be found in Figure 6.4. This annotation aids discussion of steps that adversaries may take, by allocating each part of the functional control structure a unique identifier - the aim is to reduce ambiguity when describing parts of the control loop (i.e. *MP1* is clearer than *the connection between an operator and a ground control station*).

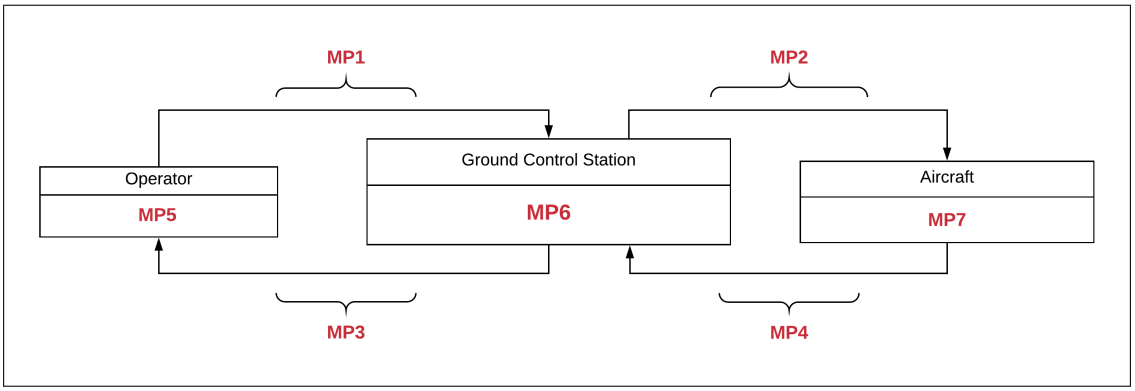

Figure 6.4: Manipulation point view of functional control structure for UAV case study.

6.2.6.2 Adversary modelling

For a system of this scale, four adversary types are considered, ranked in order of severity/threat:

1. Nation-state actors.
2. Activist/hacktivist groups and organisations.
3. Curious individuals.

4. Unintentional adversaries.

While there may be many more adversaries, four adversaries ranging so substantially in capabilities, intent and possible actions should sufficiently address any security issues inherent in the design and provide meaningful critical requirements at this stage of analysis. The benefit of the abstraction of the analysis is to highlight *architectural* flaws inherent in the design that can represent a systematic security risk, in contrast to performing a more low-level analysis which might highlight *implementation* flaws but would limit the analysis to only being valid when undertaken on a system where much of the design was already fixed.

Each adversary is presented with both a profile and (at least one) separate table representing actions that the adversary may take in order to disrupt the system. Each of these *actions* can be considered a **manipulation sequence** and will be assigned an identifier of [**MSxx**] where *xx* is a free identifier. Manipulation sequences are identified globally rather than having an identifier relative to their adversary, much in the same way that hazards are assigned a global identifier rather than one relating to their control action under analysis.

Nation-state actors: The scale of this adversary is such that all aspects of the system control structure may be under threat. See Table 6.5 for adversary profile.

Table 6.5: Nation-state actor adversary profile

Category	*Detail*
Identifier	Adversary 1 (A1)
Name/categorisation	Nation-state actors
Intent	To undermine system completely as part of a wider disruptive campaign.
Access	All manipulation points are accessible to adversary.
Information	Command language between entities is fully understood.
Actions	*Detailed in Table 6.6, Table 6.7, Table 6.8.*

Table 6.6: Nation-state adversary: Aircraft-focused adversary actions

Action name/identifier	*Action steps/explanation*
Jamming of all commands from legitimate GCS to aircraft [**MS1**]	Prevention of all messages transmitted via *MP2* and *MP4*, isolating aircraft from valid control by GCS and preventing aircraft from informing GCS of current state.
Malicious control of aircraft by malicious GCS [**MS2**]	Aircraft receive and follow commands from a malicious GCS via *MP2*, as well as providing updates to malicious GCS over *MP4*.
Permitting only selective control of aircraft by legitimate GCS [**MS3**]	Commands from GCS to aircraft via *MP2* are selectively permitted and otherwise jammed, as is feedback sent from aircraft to GCS via *MP4*.
Manipulation of control software on aircraft [**MS4**]	Aircraft are manipulated by [**MS2**] to carry malicious software unknown to legitimate GCS; this may permit the aircraft to transmit/interact with other entities maliciously with the intent of gaining a foothold into other insecure systems by the adversary.

Table 6.7: Nation-state adversary: GCS-focused adversary actions

Action name/identifier	*Action steps/explanation*
Malicious clearing of aircraft registration data [**MS5**]	GCS has registered aircraft data cleared, undermining control loop of aircraft via *MP2* and *MP4*.
Malicious clearing of plans/commands held by GCS and mapped to aircraft registrations [**MS6**]	GCS has plan & command data cleared, undermining control of aircraft via *MP2/MP4* as well as causing *operator* confusion via *MP3/MP1*.
Malicious clearing of aircraft location data held by GCS [**MS7**]	GCS has aircraft location data cleared, undermining control of aircraft via *MP2/MP4* and reducing GCS ability to meet system purpose.
Malicious adjustment of restricted/-hazardous location database [**MS8**]	GCS restricted/hazardous location database modified to either include valid/useful locations or have restricted/hazardous locations excluded, thus leading to potential aircraft loss and failure to fulfil system purpose.

Table 6.8: Nation-state adversary: Operator-focused adversary actions

Action name/identifier	*Action steps/explanation*
Malicious operator compelled to enter a large number of plans & commands into GCS [**MS9**]	A malicious operator is manipulated into entering a large number of plans & commands for validation into the GCS via *MP1*, thus resulting in a denial-of-service to other operators.
Malicious operator submits cancellations for commands belonging to other operators [**MS10**]	A malicious operator submits command cancellation requests via *MP1* for commands not belonging to them with the aim of disrupting legitimate use of the system.
Malicious operator submits commands with the intention of causing aircraft damage/denial-of-service [**MS11**]	A malicious operator submits commands via *MP1* that innately exceed safe operating conditions for the aircraft attached to the GCS with the intent of having the GCS relay these to aircraft.

Activist/hacktivist groups and organisations: This adversary is more limited in their approach and access; the relevant adversary profile can be found in Table 6.9.

Table 6.9: Hacktivist/activist adversary profile

Category	*Detail*
Identifier	Adversary 2 (A2)
Name/categorisation	Hacktivist/activist groups and organisations
Intent	To disrupt system activity as part of a political campaign.
Access	Control loop between GCS and Aircraft (consisting of *MP2, MP7, MP4*).
Information	Command language between entities not understood initially; awareness of existence of control loop consisting of *MP2, MP4*
Actions	*Detailed in Table 6.10*

Table 6.10: Hacktivist individuals/groups: All adversary actions

Action name/identifier	*Action steps/explanation*
Snooping of traffic between aircraft and GCS to increase understanding of the system and how control between the two entities operates [**MS12**]	The adversary seeks to snoop traffic between the aircraft and GCS via *MP2* and *MP4* in order to develop an understanding of control and feedback actions to facilitate future manipulations.
Transmission of recall signal to all aircraft local to adversary [**MS13**]	The adversary, based on increased system understanding developed through **MS12**, transmits the recall signal to all aircraft within range of adversary via *MP2* with the intent of disruption of the system.
Confusion of GCS through spoofing of aircraft updates/deviations [**MS14**]	The adversary transmits location updates and deviations to the GCS maliciously via *MP4* on behalf of aircraft in the area, leading to the GCS misunderstanding aircraft locations.
Transmission of update route to all aircraft to act as denial-of-service [**MS15**]	The adversary transmits update route commands via *MP2* in order to deny the legitimate use of aircraft assets. This may include sending all aircraft to one location, directing aircraft outside of GCS transmission range, etc.

Curious individuals: This category of adversary represents anyone with a non-malicious (but still potentially disruptive) interest in the system and the technical capability to potentially snoop traffic between aircraft and GCS. The adversary profile can be found in Table 6.11.

Table 6.11: Curious individual adversary profile

Category	*Detail*
Identifier	Adversary 3 (A3)
Name/categorisation	Curious individuals.
Intent	To develop an understanding of the system out of curiosity.
Access	Control loop between GCS and Aircraft (consisting of *MP2, MP7, MP4*).
Information	Command language between entities not understood initially; awareness of information being exchanged over-the-air via *MP2* and *MP4*.
Actions	*Detailed in Table 6.12*

Table 6.12: Curious individual: All adversary actions

Action name/identifier	*Action steps/explanation*
Capturing and analysis of commands and feedback exchanged by GCS and aircraft **[MS16]**	The adversary observes communication between the GCS and the aircraft via *MP2* and *MP4* and therefore develops an understanding of the control language between aircraft and GCS.
Repeating of commands sent by GCS to aircraft **[MS17]**	The adversary, in an attempt to develop further understanding of the command loop and the entities involved, chooses to capture commands sent via *MP2* and then re-transmit the captured commands over *MP2* in order to observe results in terms of feedback via *MP4*. This may result in unintended effects, such as aircraft adding the same waypoint to its route multiple times, attempting to set initial route once aircraft is already underway, etc.

Table 6.12: Curious individual: All adversary actions (continued)

Action name/identifier	*Action steps/explanation*
Repeating of feedback sent by aircraft to GCS [**MS18**]	The adversary, much like in **MS17**, chooses to capture feedback sent from aircraft to GCS via *MP4* and then repeats these captured commands via *MP4* at some later point. This can result in the GCS misunderstanding aircraft position, taking into account multiple deviations where only one has occurred, etc.

Unintentional adversaries: This class of adversary may not even be aware that the system exists - they may interact only incidentally as the communication between system entities may be similar to their own system or use similar protocols.

Table 6.13: Unintentional adversary profile

Category	*Detail*
Identifier	Adversary 4 (A4)
Name/categorisation	Unintentional adversaries.
Intent	To interact with their own systems, with incidental effect on the system under analysis.
Access	Control loop between GCS and Aircraft (consisting of *MP2, MP7, MP4*).
Information	Command language not understood; using similar protocols/transmission systems as are being used in *MP2* and *MP4*.
Actions	*Detailed in Table 6.14*

Table 6.14: Unintentional adversary: All adversary actions

Action name/identifier	*Action steps/explanation*
Disruption of control between GCS and aircraft due to traffic on similar networks/protocols **[MS19]**	Adversary makes use of similar communication protocols/networks to interact with their own systems via the same medium as *MP2* and *MP4*. The net result is aircraft and GCS receiving invalid or irrelevant commands, which can result in denial-of-service or a reduction in effective control.

6.2.6.3 Critical requirement generation

Generation of critical requirements involves the consideration of each *manipulation sequence* in order to permit a *critical requirement* to be generated in order to mitigate the identified adversarial behaviour. Each critical requirement is assigned an identifier and will later be integrated into the formal model to validate its effectiveness at eliminating the undesirable behaviour. The list of critical requirements can be found in Table 6.15.

Table 6.15: Generation of critical requirements

Manipulation sequence	*Critical requirement*
Jamming of all commands from legitimate GCS to aircraft [**MS1**]	Aircraft should attempt to complete route and return home in absence of control from GCS. [**CR29**]
Malicious control of aircraft by malicious GCS [**MS2**]	Instructions/control actions to aircraft must be accompanied by unique tokens, which are never reused. [**CR30**]
Permitting only selective control of aircraft by legitimate GCS [**MS3**]	*In addition to* ***CR29***, aircraft and GCS will maintain a message queue and will not remove items from the message queue in the absence of confirmation. [**CR31**]
Manipulation of control software on aircraft [**MS4**]	Aircraft behaving unusually will be marked as *suspect* and will not be assigned routes until their status is set to *clear* once again. Aircraft already on routes shall be permitted to complete them. [**CR32**]
Malicious clearing of aircraft registration data [**MS5**]	This will result in an inability to control aircraft, therefore, *addressed by* ***CR29***
Malicious clearing of plans/commands held by GCS and mapped to aircraft registrations [**MS6**]	*Addressed by* ***CR29***
Malicious clearing of aircraft location data held by GCS [**MS7**]	Aircraft must retain their own independent history of their locations once assigned a route to ensure aircraft location data can be reconstituted. [**CR33**]

Table 6.15: Generation of critical requirements (continued)

Manipulation sequence	*Critical requirement*
Malicious adjustment of restricted/-hazardous location database [**MS8**]	Where aircraft hazardous/restricted location database is greater in size than that of the GCS, the aircraft database is to be retained [**CR34**].
Malicious operator compelled to enter a large number of plans & commands into GCS [**MS9**]	Operators are limited to submitting up to 10 plans/commands at any time [**CR35**]
Malicious operator submits cancellations for commands belonging to other operators [**MS10**]	Operators may not cancel the plans of other operators [**CR36**]
Malicious operator submits commands with the intention of causing aircraft damage/denial-of-service [**MS11**]	*Partially addressed by* ***CR35***; GCS may not include waypoints for aircraft outside of maximum control range [**CR37**]
Snooping of traffic between aircraft and GCS to increase understanding of the system and how control between the two entities operates [**MS12**]	Traffic between aircraft and GCS should be obfuscated [**CR38**]
Transmission of recall signal to all aircraft local to adversary [**MS13**]	*Addressed by* ***CR30***
Confusion of GCS through spoofing of aircraft updates/deviations [**MS14**]	Feedback from aircraft to GCS should be accompanied by unique tokens, which are never reused [**CR39**]
Transmission of update route to all aircraft to act as denial-of-service [**MS15**]	*Addressed by* ***CR30***
Capturing and analysis of commands and feedback exchanged by GCS and aircraft [**MS16**]	*Addressed by* ***CR38***
Repeating of commands sent by GCS to aircraft [**MS17**]	*Addressed by* ***CR30***
Repeating of feedback sent by aircraft to GCS [**MS18**]	*Addressed by* ***CR39***
Disruption of control between GCS and aircraft due to traffic on similar networks/protocols [**MS19**]	Aircraft and GCS will only process messages/instructions that are correctly formatted and expected [**CR40**]

6.2.7 Step 7 - Integration of critical requirements into formal model

The integration of the critical requirements into the formal model occurs in multiple refinements; detail of each refinement can be found in Table 6.16.

Table 6.16: Model refinements

Model refinement	*Detail*
Initial formal model [**MR0**] Full model available in Appendix C	Formal model created during Step 4 of analysis subsection 6.2.4. Represented within DroneMachineInitial and DroneContextInitial files.
Model refinement 1 [**MR1**] Full model available in Appendix D	First refinement - represented within DroneMachine2 and DroneContext2 files.
Model refinement 2 [**MR2**] Full model available in Appendix E	Second refinement - represented within DroneMachine3 and DroneContext3 files.
Model refinement 2 [**MR3**] Full model available in Appendix F	Third refinement - represented within DroneMachine4 and DroneContext4 files.

The process of integrating the critical requirements into the formal model can take many forms within the model itself. Detail of each critical requirement against details of how it was integrated into a refinement of the formal model is available in Table 6.17.

In addition to this, it is important to note that it may not be possible to integrate some critical requirements into the formal model as refinements where these critical requirements represent substantial design changes to the underlying system as the formal model is based on the system as scoped in subsection 6.2.1. This is a limitation of the formal model as substantial changes to the design may be incompatible in terms of formal representations when compared to the initial system formal model, and so it is not possible to vertically refine the model to accommodate the change. More detail on how critical requirements that necessitate such substantial design changes are handled can be found in subsection 6.2.9.

Table 6.17: Detail of integration of critical requirements into formal model

Critical Requirement	*Model refinement/details*
Transmitted plans will be entered into a validation queue and shall be validated in order of submission. [**CR1**]	Added in **MR1** through introduction of variables and invariants relating to plan and command validation queues, and the addition of guards and actions to relevant events such as TransmitCommand & ValidateCommand as two examples.
Commands must not contain unsafe waypoints. [**CR2**]	Initially in place in **MR0** when adding a new restricted location results in all commands containing that location being marked as *invalid* and aircraft assigned said command being commanded to return home. Expanded in **MR1** through guards on TransmitCommand to ensure commands contain no unsafe waypoints.
Transmitted commands will be entered into a validation queue and shall be validated in order of submission. [**CR3**]	Added in **MR1** similarly to how **CR1** was integrated.
Cancellation of a command must specify a command which already exists. [**CR4**]	Initially in place with **MR0**, CancelCommand event requires specified command to exist within the set modelling all transmitted commands.
Cancellation may only occur for commands which are not already complete [**CR5**].	Integrated in **MR2**, guards on CancelCommand event check for the status of the specified command to ensure the command is not already complete.
Plan validation states must always be reported accurately. [**CR6**]	Modelled from **MR0** where the QueryPlanValidationState event reports the plan validation state.
Plan validation state must always be reported when requested. [**CR7**]	Modelled from **MR0** where the QueryPlanValidationState will always report the plan validation state when requested.
Plan validation state must always be reported as *undetermined* if requested prior to validation being complete. [**CR8**]	Transmitted plans are set to *undetermined* validation state once received and this behaviour is modelled from **MR0**.

Table 6.17: Detail of integration of critical requirements into formal model (continued)

Critical Requirement	*Model refinement/details*
Command validation state must always be reported accurately. [**CR9**]	Modelled from **MR0** where the QueryCommandValidationState event reports the command validation state.
Command validation state must always be reported when requested. [**C10**]	Modelled from **MR0** where the QueryCommandValidationState event reports the command validation state.
Command validation state must always be reported as *undetermined* if requested prior to validation being complete. [**CR11**]	Transmitted commands are set to *undetermined* validation state once received and this behaviour is modelled from **MR0**.
Command cancellation confirmation is not to be issued until aircraft has ceased following route. [**CR12**]	Modelled in **MR0**, command cancellations via the CancelCommand event are viewed to be atomic and so aircraft are instantly assigned new routes as soon as the cancellation is issued. The ConfirmCancellation event will return TRUE if specified command is marked as cancelled.
Command cancellation state must always be reported. [**CR13**]	Modelled from **MR0**. Due to the nondeterminism of the model, command cancellation state will be *eventually* reported.
Command cancellation confirmation must only ever be issued in response to a cancellation request. [**CR14**]	It is only possible to confirm cancelled commands once the command has been cancelled so this is modelled from **MR0**.
Initial route must not contain restricted or hazardous waypoints. [**CR15**]	TaskAircraft event ensures this is not the case via a guard on the event checking the command validation state from **MR2**. Furthermore, the registration of a new restricted location from **MR0** ensures aircraft have their routes replaced if the existing route becomes hazardous.

Table 6.17: Detail of integration of critical requirements into formal model (continued)

Critical Requirement	*Model refinement/details*
Commands passing validation must be passed to aircraft in the form of routes. [**CR16**]	Due to the non-deterministic nature of Event-B models, this is hard to model. TaskAircraft will *eventually* pass all valid commands to aircraft that have no route assigned from **MR2**.
Aircraft may not have their initial routes set to commands that do not pass validation [**CR17**]	Modelled successfully from **MR2** using a guard in TaskAircraft that checks command validation state.
Updates provided to aircraft may not contain restricted or hazardous waypoints. [**CR18**]	Modelled from **MR2** using a guard in RetaskAircraft that checks command validation state.
Routes that aircraft are currently on should be updated to remove hazardous waypoints or return aircraft home. [**CR19**]	Modelled from **MR0** when registration of a new restricted location will cause the aircraft route to be replaced with a route to return home.
Aircraft may not be updated until provided with an initial route [**CR20**]	Modelled from **MR2** where guards on RetaskAircraft will check that the route for an existing aircraft is not blank.
Aircraft may not be recalled when to do so is hazardous [**CR21**]	This was not possible to model and is therefore potentially a poor quality critical requirement.
Aircraft may only be recalled while on a route [**CR22**]	Modelled from **MR0** - aircraft may not be recalled when at home, and therefore will always be on a route.
Aircraft may only confirm receipt of update/initial route when provided one by the GCS. [**CR23**]	Modelled from **MR2**, ConfirmReceipt event guards against confirmation happening without route being assigned. Confirmations may only occur once.
Aircraft will always confirm receipt of update/initial route when one is provided by the GCS. [**CR24**]	Modelled from **MR2** - once more, due to the non-determinism of the underlying model, this will eventually occur but it cannot be guaranteed immediately.

Table 6.17: Detail of integration of critical requirements into formal model (continued)

Critical Requirement	*Model refinement/details*
Aircraft must report accurate, regular location data. [**CR25**]	Modelled from **MR2**, where multiple events (ReportLocationElsewhere, ReportLocationHomeNormal, ReportLocationHomeRecall) cover all possibilities of airports reporting location data.
Aircraft must report locations with a time-stamp to prevent sequencing issues. [**CR26**]	Modelled from **MR1** where locations must be reported in a strictly-increasing manner.
Aircraft must report accurate deviations where these occur. [**CR27**]	Modelled most completely from **MR2**, where deviations are reported so long as they do not belong to the existing route.
Aircraft must report deviations with a time-stamp to prevent sequencing issues. [**CR28**]	Modelled from **MR1** where deviations must be reported in a strictly-increasing manner.
Aircraft should attempt to complete route and return home in absence of control from GCS. [**CR29**]	Modelled from **MR2** where aircraft will report locations in sequence and then return home automatically. It is however difficult to model the notion of *removing* the GCS control from the model.
Instructions/control actions to aircraft must be accompanied by unique tokens, which are never reused. [**CR30**]	Modelled from **MR3** through extensive guards and actions added to events, and the maintenance of a UsedTokens variable, as well as a TokenMapping between Aircraft and Tokens. A lone token is removed for each action involving the GCS communicating with the aircraft.
Aircraft and GCS will maintain a message queue and will not remove items from the message queue in the absence of confirmation. [**CR31**]	*Not possible with formal model in present state.*

Table 6.17: Detail of integration of critical requirements into formal model (continued)

Critical Requirement	*Model refinement/details*
Aircraft behaving unusually will be marked as *suspect* and will not be assigned routes until their status is set to *clear* once again. Aircraft already on routes shall be permitted to complete them. [**CR32**]	Modelled from **MR3** - aircraft may be marked suspect at any time, will complete their routes and may not be tasked/re-tasked until they have both reached the home location and had their status set back to normal.
Aircraft must retain their own independent history of their locations once assigned a route to ensure aircraft location data can be reconstituted. [**CR33**]	*Not possible with formal model in present state.*
Where aircraft hazardous/restricted location database is greater in size than that of the GCS, the aircraft database is to be retained [**CR34**].	*Not possible with formal model in present state.*
Operators are limited to submitting up to 10 plans/commands at any time [**CR35**]	Modelled from **MR3** - a given operator may only have 10 of each plans and commands and this is enforced through invariants and guards on all relevant events.
Operators may not cancel the plans of other operators [**CR36**]	Modelled from **MR3** - CommandOwnership variable maps Operators to Commands and guards in CancelCommand prevent the cancellation of a command owned by another operator.
GCS may not include waypoints for aircraft outside of maximum control range [**CR37**]	*Not possible with formal model in current state.*
Traffic between aircraft and GCS should be obfuscated [**CR38**]	*Not possible with formal model in current state.*

Table 6.17: Detail of integration of critical requirements into formal model (continued)

Critical Requirement	*Model refinement/details*
Feedback from aircraft to GCS should be accompanied by unique tokens, which are never reused **[CR39]**	Modelled from **MR3** through extensive guards and actions added to events, and the maintenance of a UsedTokens variable, as well as a TokenMapping between Aircraft and Tokens. A lone token is removed for each action involving the aircraft communicating with the GCS.
Aircraft and GCS will only process messages/instructions that are correctly formatted and expected **[CR40]**	*Not possible with formal model in current state.*

The statistics relating to proof obligations in each refinement can be found in Table 6.18.

Table 6.18: Statistics on proof obligations for each model refinement

Refinement	*Total*	*Automatically proved*	*Manually proved*	*Undischarged*
MR0	30	29	1	0
MR1	22	9	13	0
MR2	89	58	31	0
MR3	76	52	17	7[2]

6.2.7.1 Challenges of representing critical requirements within the formal model

Representing the critical requirements within the formal model was a task for which many aspects of the Rodin toolset were utilised. Examples are given below, demonstrating challenges of integrating critical requirements, as well as the tool support that was used to resolve these challenges. These challenges arose due to the complexity of translating natural language requirements into formal aspects of a system model and further demonstrated that the natural language critical requirements, while often sensible and

[2]This is due to the proof obligations around ensuring that operators may not submit more than 10 plans and commands for validation at any point which the author has been unable to discharge.

detailed, still fell short of the formalism and detail required to automatically translate into constraints on the system model.

Validation queues and sequencing The introduction of *CR1* into the formal model occurred during **Model Refinement 1** and involved the creation of queues for both commands and plans, in addition to a counter which tracked the number of items in the respective queues. This enabled plans to be submitted into queues, where they would be mapped (via a partial injection) to a value in the range of $0 \ldots PlanValidationQueueCount-1$. A similar mechanism exists for commands. The intent of this implementation was to model a first-in first-out (FIFO) queue structure.

In order to satisfy the relevant invariants, this meant that all events which involved adding to the validation queues needed to achieve two things:

1. To modify the relevant queue variable through adding an item to the queue and mapping it to the current value of the counter variable.
2. To incremeent the relevant counter variable to ensure it reflected the number of items within the queue after the action had been taken.

Events that sought to remove from the queue (i.e. the validation events) had to undertake the above two actions (albeit from the opposite perspective of shrinking the queue and decrementing the counter variable), in addition to the following:

3. Iterating through the current queue and adjusting all of the mappings to their new positions.
    - This was due to the fact that the former 'first' item in the queue had been removed, which would make the new first item in the queue map to the value 1. However, the first item in the queue should map to the value 0, as this was the position that removal events accessed.

The desired functionality was achieved using a lamba operator and the set comprehension operator. However, this was relatively complex for the author who had no prior experience with either of these operators, and so the Rodin tool had several aspects of functionality which aided with the implementation of this critical requirement:

1. **Animation** was used extensively. This involved the manual choice of a sequence of events which exercised the queue functionality (i.e. *TransmitPlan* followed by *ValidatePlan*). The ability to inspect the state of the machine at all stages enabled the actions to be reworked until they functioned correctly in terms of a FIFO queue.

- This was followed by a number of *Random Animation* runs, where the traces and model state were inspected at each step to ensure that the queues behaved correctly at each step.

2. **Proof obligations** were generated for the invariants which provided the *type* for *PlanValidationQueue* and *CommandValidationQueue*. As both of these variables were typed to map from a plan/command to a value in the range of $0 \ldots PlanValidationQueueCount - 1$, this provided an additional level of scrutiny over the behaviour of events that interacted with the queue:

   - An action attempting to map above that range (i.e. mapping to PlanValidationQueueCount without the subtraction due to zero-indexing) or below the valid range (i.e. -1) would result in the proof obligation remaining undischarged.
   - Furthermore, the violation of the invariant during animation provided a counter-example, as well as all relevant state information, to enable the action to be corrected.

3. Once a high degree of confidence was had regarding the behaviour of the FIFO queues for plans/commands, Rodin's model checking was utilised and ran for several hours in various configurations to check whether any invariant violations could be found. This did not highlight any violations within its execution[3].

The process of modelling this critical requirement highlighted that the wording of the critical requirement could have been improved; the wording 'shall be validated in order of submission' could have been substituted for something more prescriptive such as 'shall be entered into a first-in, first-out (FIFO) queue' which would've focused modelling activities accordingly.

AircraftRoutes improvements This improvement occurred before critical requirements integration could begin, and was due to a deficiency identified in the initial formal model. In the initial formal model, the *AircraftRoutes* variable mapped the *RegisteredAircraft* variable to the *Commands* variable, however, this resulted in residual proof obligations which could not be discharged, and frequent violation of invariants during animation.

Further investigation indicated that this was due to the need for aircraft to possess an empty route for times when they were not on a route (i.e. a route consisting of $\varnothing$), as well as a route for when they were returning home due to being recalled, or being at the end of their route (i.e. a route consisting of only the HOME location). This resulted in

[3]Due to the search space, this is not a hard guarantee, but it provided more confidence that the invariants and actions involved in the queueing behaviour were correct.

the expansion of the AircraftRoutes type invariant to map from the *RegisteredAircraft* variable to Commands $\cup \{\text{HOME}\} \cup \{\varnothing\}$.

This permits a registered aircraft to have a normal route, a route just consisting of 'HOME' or, when at the home location, to have no route at all.

Functions and relations The choice of functions and relations to model new variables where these variables are used to map between existing variables proved quite complex on a case study of this scale. However, this choice paid substantial dividends in terms of ensuring that the behaviour of the model was correct with regards to the critical requirements, as the characteristics of the functions and relations being violated generated useful counter-examples and proof obligations that could not be discharged. Several examples are given below:

1. The use of a partial injection as part of the Command and Plan queue variables ensured that there was not any duplicate mapping to positions within the queue (i.e. no two elements shared an index of 0 at any one time). The violation of this invariant by several events drew attention to the actions involved and enabled them to be corrected.
2. The use of a number of total functions within a number of mapping variables highlighted that the registration of aircraft did not have actions corresponding to some variables. This was corrected as otherwise proof obligations relating to these variables could not be discharged.
3. The animation of the model often revealed that the 'wrong' relation or function had been used in several places. This encouraged the use of a more-specific function/relation, which then highlighted further events that were not behaving correctly with regards to all variables.

Broadly speaking, the majority of undischarged proof obligations often reflected that events were not meeting their obligations as required by the critical requirements. The use of specific types of relations/functions enabled these issues to be detected and corrected, and to ensure that all events interacting with these variables behaved correctly and in a well-defined manner.

ReportLocation refinement The refinement of the *ReportLocation* event in **Model Refinement 2** involved the creation of three events which all extended from the abstract *ReportLocation* event in *DroneMachine2*:

- *ReportLocationElsewhere*, for aircraft to report when they are anywhere aside from home (i.e. when they are in transit and following a route).

- Two events were then required to adequately model the act of an aircraft returning home, as there are two situations in which this can occur:

    1. *ReportLocationHomeNormal*, which reports the aircraft returning to the *HOME* location at the end of a route.
    2. *ReportLocationHomeRecall*, which reports the aircraft returning to the *HOME* location in response to the aircraft being recalled by the Ground Control Station.

The rationale for the different events for reporting when an aircraft returns to the *HOME* location 'normally' versus when it is recalled relates to other variables within the model which are set when the *RecallAircraft* event is triggered. In order to clear these, the *ReportLocationHomeRecall* adjusts several of these variables back to states which indicate that the aircraft has been successful recalled. In contrast, the *ReportLocationHomeNormal* serves to essentially reset the aircraft to the 'default' state that a newly registered aircraft would be in. These behaviours did not require modifications to *ReportLocationElsewhere* as recalled aircraft essentially terminate their existing route and are set to only return home, so the behaviour of *ReportLocationElsewhere* is consistent in all possible situations as there is a guard to prevent recalled aircraft utilising this event.

This combination of events required extensive testing to ensure that the model behaved appropriately and did not enter into nonsensical states (e.g. an aircraft being recalled and triggering the *ReportLocationHomeNormal* event). This involved two steps:

1. The first step involved the creation of a number of guards (e.g. on the relevant events and the discharging of proof obligations as a number of the guards had well-definedness proof obligations due to being non-trivial predicates.
2. The second step involved animation which was undertaken manually (via manual choice of enabled events) and then automatically (via Random Animation) to ensure that aircraft were triggering the appropriate event for their situation.

Undertaking this refinement demonstrated how a number of critical requirements influencing the same system behaviour can complicate the formal model, but the tools provided by the Rodin platform can be used to ensure that complex behaviours such as these can be demonstrated to be correct.

6.2.8 Step 8 - Causal factors analysis

At this stage, the bulk of critical requirements that can be integrated into the formal model have been, thus increasing the assurance that these critical requirements are meaningful. However, the process of identifying hazards in the control action analysis

step (subsection 6.2.5) does not meaningfully tell us how such hazards can arise, simply that they might as a result of the standard analysis table.

To bolster this aspect of the analysis, SE-STPA uses the existing process of *causal factors analysis* from the original STPA process. This takes each hazard identified in the control action analysis and seeks to understand *how* it might arise within the context of the control loop in which it takes place. This aids the identification of potential *human factors* issues or areas where the process models of controllers may not be consistent. These issues can all result in the hazardous behaviour arising in the first place, and therefore overt identification and generation of critical requirements can once again ensure the system does not enter a hazardous state.

In the interest of brevity, a select number of causal factor analyses are presented in this subsection relating to the UAV case study to demonstrate the value of this step of analysis.

6.2.8.1 Existing route becomes hazardous in some way and is not corrected by an update

This hazard relates to the *Update Route* control action not being issued in response to a material change in circumstances that makes an aircraft's existing route hazardous in some way. In line with the system-level hazards identified in subsection 6.2.1, this would put the system into a hazard state represented by **Hazard 4** and/or **Hazard 5**.

As the responsibility for tasking aircraft and ensuring they follow safe routes lies with the Ground Control Station, and operators have no direct influence on aircraft once they are following a route, the control structure relevant to this hazard is only the control loop between the GCS and an aircraft.

The causal factors associated with the hazard can therefore be found in Figure 6.5.

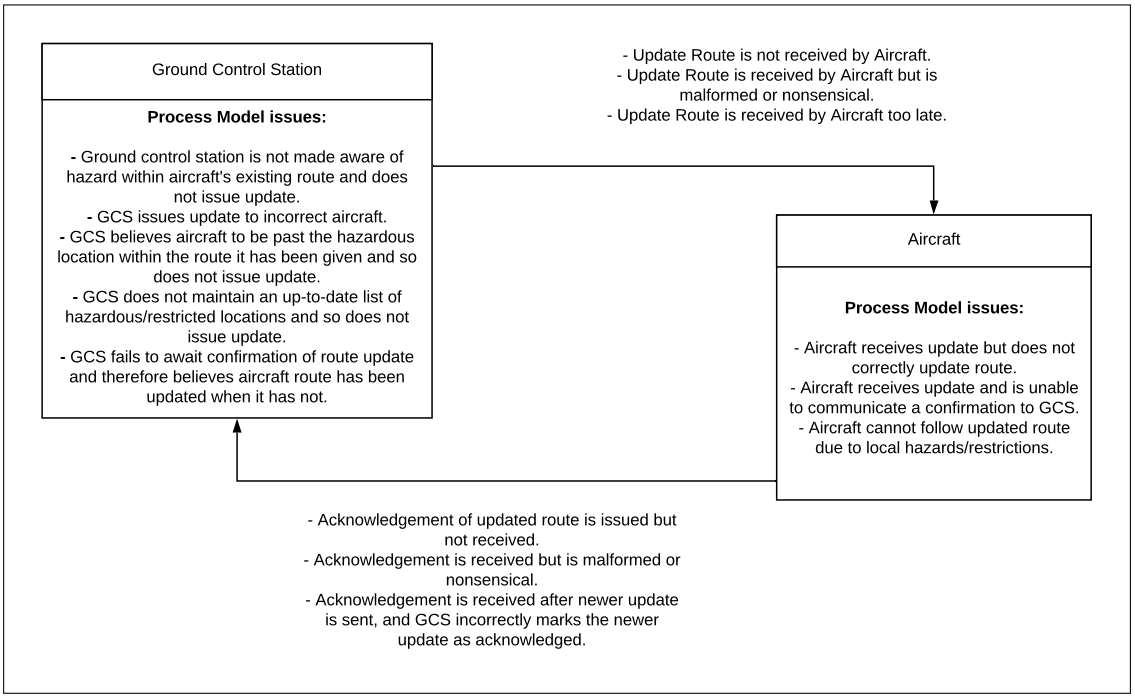

Figure 6.5: Causal factors breakdown for Failure-to-Update-Route Hazard

From this, several observations can be made:

1. Several of the identified causal factors relate to the notion of whether commands and confirmations between the GCS and aircraft are received and processed correctly by both entities. This will substantially be mitigated by **CR31** when this critical requirement is integrated into a future design iteration of the system.

2. Several other causal factors relate to the notion of information exchanged by both parties becoming malformed in transit - this will be mitigated by **CR40** being integrated into the design.

3. One further identified causal factor is around timeliness and is mitigated by **CR26**, although the integration of this critical requirement to this version of the model is partial and could be more thoroughly integrated into a future system design by modelling the passage of time. This would enable the locations to be reported with timestamps and model more-concrete functionality of the system.

The remainder of causal factors rely on a failure of either the GCS or the aircraft to act in line with their process models, and this can be mitigated against by the following design revisions/critical requirements:

CR41: Aircraft shall attempt to follow updates even if unable to communicate a confirmation back to the GCS.

CR42: Aircraft are permitted to skip hazardous stops on assigned routes if they are unable to navigate them safely.

CR43: Aircraft will notify of skipped stops, as defined in **CR42**, as deviations to the GCS as soon as they are able.

This covers the identified causal factors for this hazard, and provides further critical requirements for refining the design when iterating or re-scoping.

6.2.8.2 Initial route is set before command passes validation

This hazard relates to the *Set initial route* control action being issued **prior to** a command being successfully validated and therefore being safe to pass to aircraft. In line with system-level hazards identified in subsection 6.2.1, the risks placing the system into a hazard state defined by **Hazard 4**, **Hazard 5** and **Hazard 6** or some combination thereof.

This is once again an issue that is essentially contained purely to the control loop between the GCS and an aircraft, therefore the functional control structure to be considered can be isolated to these two entities. The causal factors diagram can be found in Figure 6.6.

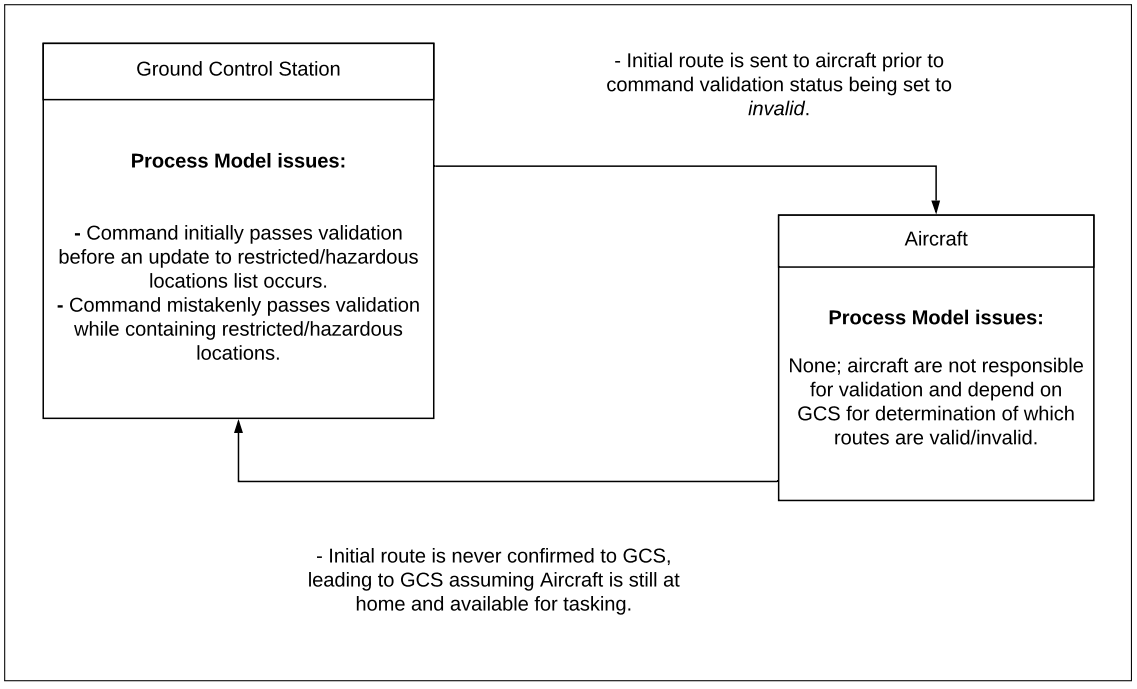

Figure 6.6: Causal factors breakdown for Initial-Route-Potentially-Invalid Hazard

Observations can once again be made based off of this diagram:

1. The bulk of causal factors here have been identified and mitigated by both **CR15**[4] and **CR17**[5] when this hazard was first identified in subsection 6.2.5.

2. The causal factor around the aircraft failing to transmit the confirmation of initial route, is mitigated by **CR24**[6] as well as **CR31**[7].

3. The final causal factor (a command is validated but then subsequently should become invalidated) is mitigated by **CR19**[8].

As there are no more causal factors that do not already have critical requirements, this means there are no further critical requirements that can be derived from this hazard as a result of causal factors analysis.

6.2.8.3 Command validation state is reported incorrectly

This hazard relates to the *Report Validation State of Command* control action reporting the validation status for a given command, but doing so incorrectly. This can lead to operators assuming an aircraft will be tasked to carry out a command when the command has failed validation, or believing that a command has failed validation when in fact it has passed.

This issue spans the entirety of the control loop across the system as the GCS reports command validation state to operators, but will also be interacting with aircraft potentially if the command has actually passed validation.

Two cases can be considered in terms of causal factors (and to keep the diagram from becoming too crowded) - the one in which the state is reported to be *valid* when it is in fact *invalid*, and the other when the command validation state is reported as *invalid* when it is in fact *valid*.

[4] Initial route must not contain restricted or hazardous waypoints

[5] Aircraft may not have their initial routes set to commands that do not pass validation

[6] Aircraft will always confirm receipt of update/initial route when one is provided by the GCS.

[7] Aircraft and GCS will maintain a message queue and will not remove items from the message queue in the absence of confirmation.

[8] Routes that aircraft are currently on should be updated to remove hazardous waypoints or return aircraft home.

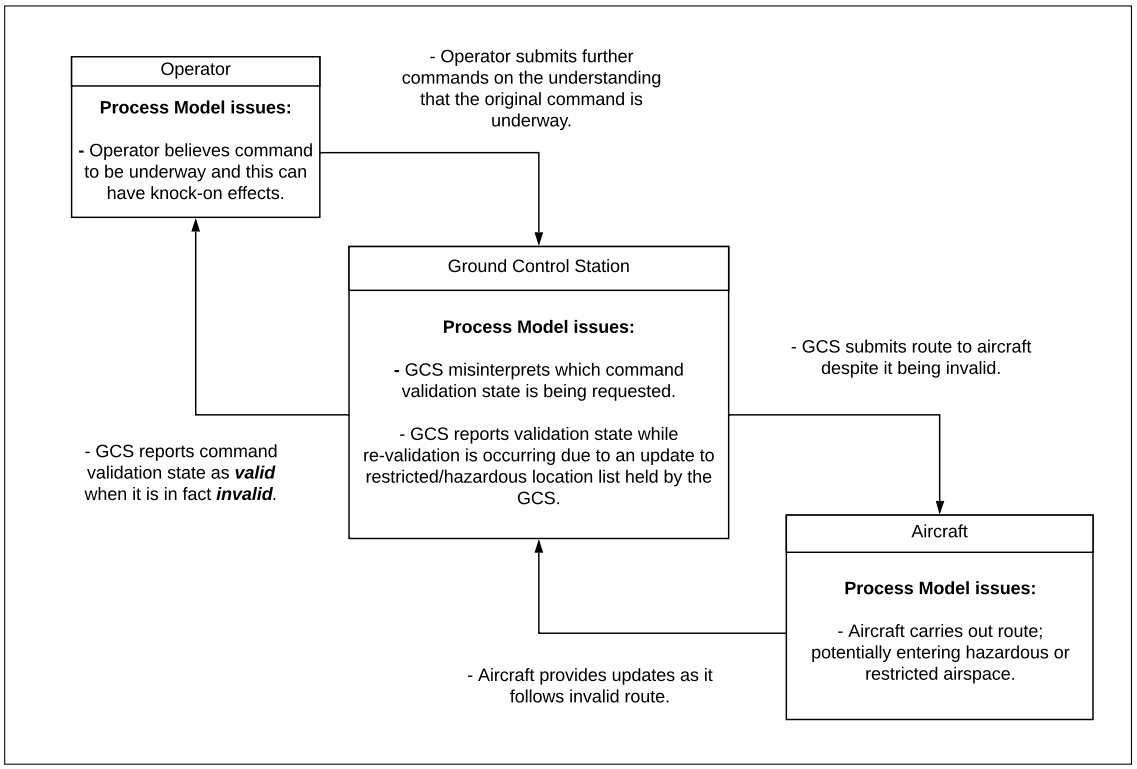

Figure 6.7: Causal factors breakdown for Command-Validation-Incorrectly-Reported Hazard - Approach 1

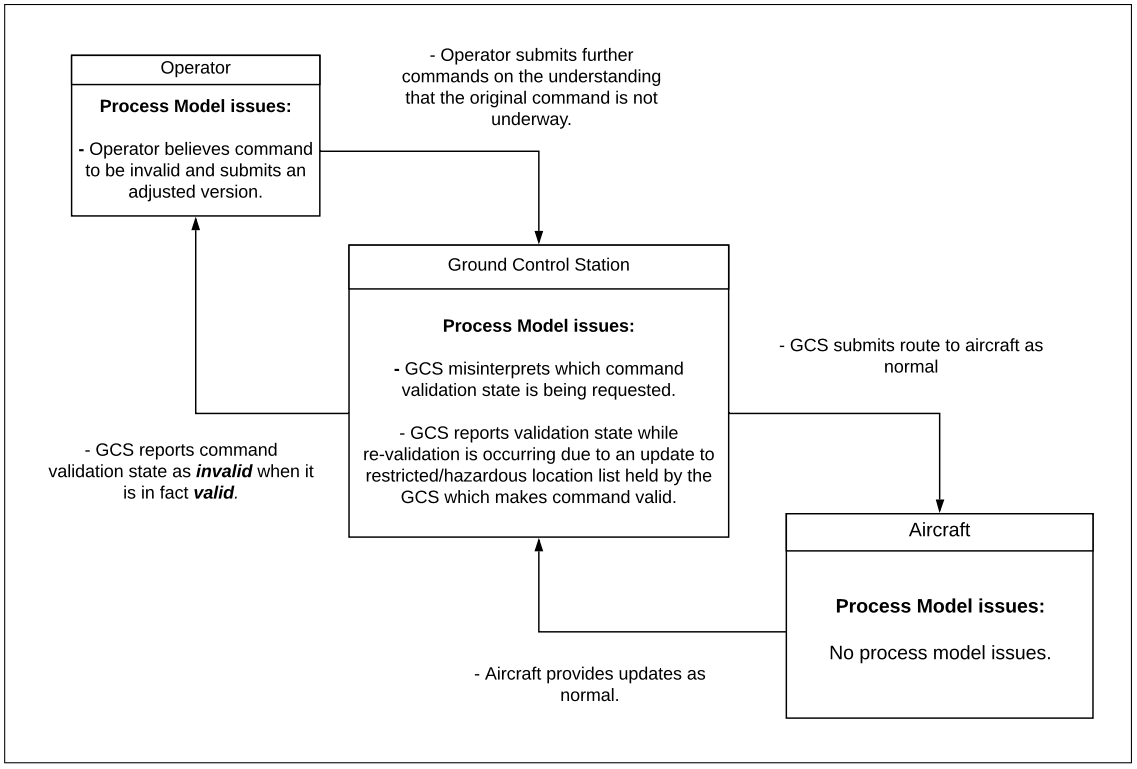

Figure 6.8: Causal factors breakdown for Command-Validation-Incorrectly-Reported Hazard - Approach 2

Observations can be made for both cases:

1. The causal factor around misinterpreting which command validation state is being requested can be resolved by ensuring that operators interfaces are clear in displaying the status of their submitted commands and the current validation state.
2. Commands undergoing a process of re-validation should be set back to *undetermined* prior to the re-validation occurring in order to prevent validation state from being reported incorrectly, therefore preventing this causal factor from being an issue. Aircraft on a route that is being revalidated should be recalled in the interest of not further violating system-level hazards.
3. The remaining causal factors appear to be a result of the natural functioning of the system and do not warrant further constraints.

This permits the creation of three further critical requirements:

CR44: Upon an update to restricted/hazardous locations list, any commands affected will be set back to *undetermined* validation status until re-validated.

CR45: Aircraft following commands that are set back to undetermined must be recalled immediately.

CR46: Operators interfaces must clearly show all command validation statuses to prevent potential confusion.

CR44 and **CR45** essentially represent a *refinement* of the existing critical requirements governing the interaction of command validation states, hazardous/restricted locations and when aircraft can be tasked with commands. These existing critical requirements (**CR15** through **CR19**) should therefore be read in conjunction with the newly-created critical requirements and will be handled accordingly when refining/revising the system in subsection 6.2.9 to ensure they do not conflict.

6.2.9 Step 9 - Iteration of design & further analysis

This step of analysis allows the design of the UAV system to be iterated with the full benefit of the critical requirements derived during the preceding steps. This should result in a design that is more safe and secure, and is somewhat more concrete than the original system design.

Several of the critical requirements proposed design modifications which should be taken into consideration at this stage of the analysis; these design changes are distilled (against the relevant critical requirements) in Table 6.19.

Table 6.19: Design modifications resulting from critical requirements

Critical Requirement	*Design modification*
CR45: Aircraft following commands that are set back to undetermined must be recalled immediately.	Aircraft shall have the ability to support an 'emergency recall'. Aircraft shall return immediately to their home location in response to an emergency recall message from the GCS.
CR31: Aircraft and GCS will maintain a message queue and will not remove items from the message queue in the absence of confirmation.	Aircraft have a distinct message queue. Outgoing confirmations and other messages shall be placed into this queue. Items shall not be removed from the queue until confirmation is received from the GCS. The GCS shall equally have its own message queue per aircraft. Outgoing messages, commands and updates shall be placed into this queue. Items shall not be removed from the queue until confirmation is received from the relevant aircraft.
CR34: Where aircraft hazardous/restricted location database is greater in size than that of the GCS, the aircraft database is to be retained	The process for updating hazardous/restricted location database will involve a reconcilation process to ensure that both the GCS and aircraft are aligned on where hazardous locations are.
CR34: Aircraft must retain their own independent history of their locations once assigned a route to ensure aircraft location data can be reconstituted	Aircraft shall maintain their own location log.
CR37: GCS may not include waypoints for aircraft outside of maximum control range	GCS shall define a maximum control range, and shall check that a route contains no waypoints outside of this control range.
CR38: Traffic between aircraft and GCS should be obfuscated	The communication channel between aircraft and GCS shall utilise reasonable security features so as to ensure that communication is not in cleartext.
CR40: Aircraft and GCS will only process messages/instructions that are correctly formatted and expected	Messages shall have a data integrity field which will be checked by the receiving element of the system.

The remaining body of critical requirements, as they refine existing behaviour of the system, shall also be carried forward as part of the design. A new functional control structure capturing the design modifications can be found in Figure 6.9. This represents a solid starting point for a second round of analysis.

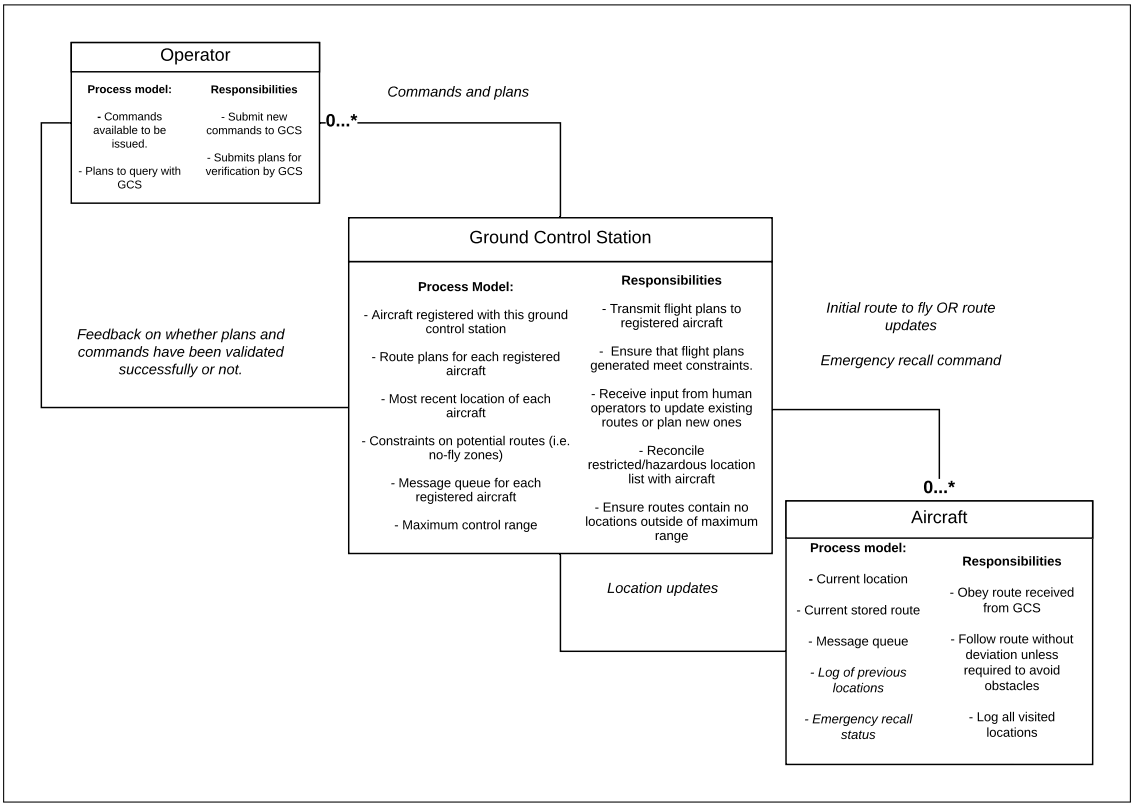

Figure 6.9: Modified functional control structure to account for design changes

6.3 Lessons learned

The lessons learned from undertaking this case study with the third version of methodology are as follows:

- Adversarial modelling represents an explicit identification of security threat actors. It can serve as the basis for thoughtful discussion about the ways in which the system can be manipulated by an adversary, and can be used to generate critical requirements to mitigate or reduce the attack surface of the cyber-physical system in question. This step is more robust than the previous attempts to concentrate the majority of security analysis into causal factors analysis.
- The structured usage of the formal model at steps 4 and 7 of the methodology requires that critical requirements are generated that are specific enough to mitigate against their identified hazards. Without the formal method, it is likely that

critical requirements may end up being vague and difficult-to-implement, which might result in system behaviours which are poorly defined or counter-intuitive when compared to the critical requirement.

- Security issues are considered at multiple stages of the analysis which ensures that security and safety hazards are given equal treatment and consideration within the analysis process. This is a significant improvement from the second iteration of the methodology, where security ended up being left to the 'end' of the analysis in the causal factors analysis step.
- The methodology in its current state can be applied to a cyber-physical case study of reasonable size and draw out a useful number of critical requirements, which can then be utilised to refine and improve on the design of the system. Though this case study does not go through a second analysis loop, it is clear that integration of a number of critical requirements into the design will result in a substantially safer and more secure system.

There are also a number of improvements which could be taken forward into a future version of the methodology:

- In order to improve the robustness of the methodology, traceability should be stipulated between stages and a clear expectation should be communicated within the description of the methodology as to the appropriate level of traceability to be expressed at each stage. The current version of the methodology does not make explicit the *minimum* traceability which should exist between artefacts of the analysis and so it can be unclear if all hazards have been adequately mitigated.
- The notion of 'trust' between entities of the system is currently not modelled. This can make a difference in both security and safety terms for the system; as a simplistic example, a system may consist of a number of sensors which are used in a voting configuration, either for redundancy or security reasons. Future versions of the methodology may wish to explore how this is modelled if it is key to system behaviour and how it is analysed for possible safety/security impacts.

Chapter 7

Discussion

This chapter begins with a restatement of the research questions before exploring the contribution of this thesis towards each of these research questions. It additionally explores the shortcomings of SE-STPA.

7.1 Research questions

The intent of this thesis was to provide answers to the following three research questions:

RQ1: Can a methodology be produced to analyse the requirements and design of cyber-physical systems with an aim to improve both their security and safety?

RQ2: Can the methodology additionally leverage formal method techniques in order to provide assurance that the mitigations generated as a result of analysis are demonstrably successful in reducing or eliminating security and safety hazards?

RQ3: Can the methodology demonstrate its utility through application to case studies of differing sizes and complexity, ideally utilising case studies involving cyber-physical systems in multiple domains?

The findings, implications and limitations against each research question can be found in the following sections.

7.2 Discussion of Research Question 1

Can a methodology be produced to analyse the requirements and design of cyber-physical systems with an aim to improve both their security and safety?

7.2.1 Context and approach

The approach to addressing this research question was to take an existing methodology with a significant history and reputation within one of the relevant domains and then modify it to be able to undertake analysis on the other domain, in a unified manner. The chosen methodology was Leveson's Systems-Theoretic Process Analysis (STPA) and its associated Systems-Theoretic Accident Model and Processes (STAMP). This existing work has been used extensively with regards to carrying out safety analysis across a multitude of domains (as detailed in Section 3.3) and therefore the intent of this thesis was to modify STPA to enable it to undertake security analysis. This approach enabled the author to build upon the existing documentation and case studies that exist from applying STPA to a number of domains.

The drivers behind the choice of this methodology are provided in Section 3.2. The approach to this research question also sought to address a number of shortcomings identified in existing attempts to utilise STPA, which has been detailed in Section 3.5, and in particular reference to the shortcomings identified when STPA is used for security analysis as detailed in subsection 3.5.3.

7.2.2 Contribution

The modification of STPA and STAMP to account for security occurred in two major steps:

1. Development of the theoretical model known as *Systems-Theoretic Accident & Attack Model and Processes* (STAAMP) from the existing Systems-Theoretic Accident Model & Processes (STAMP) model.
2. Development of the Security-Enhanced Systems-Theoretic Process Analysis (SE-STPA) to accomplish co-analysis of both security and safety issues in a single methodology.

Each of these is therefore considered as their own distinct contribution in answering this research question, and can be found in the following sections.

7.2.2.1 Contribution of the STAAMP theoretical model

Conceptualising safety and security together The theoretical model proposed by this thesis reconciles safety and security within a systems-theoretic model of system behaviours. It expands on the view proposed by Leveson in her STAMP model, which conceptualised safety in terms of Rasmussen's 'boundary of acceptable performance'

by including a second boundary representing the dividing line between a system being secure and a system entering into an insecure state. This second boundary is not parallel to that of the safety boundary and intersects it at a single point, representing the notion that there are choices or decisions which can be made in the context of the system which can degrade both security and safety. Equally, however, there remain choices which can improve safety and security jointly, and decisions which can improve safety but degrade the security of a system and vice-versa. A visual representation of the boundary of security & safety concept is given in subsection 3.7.1 and is repeated in this section as Figure 7.1.

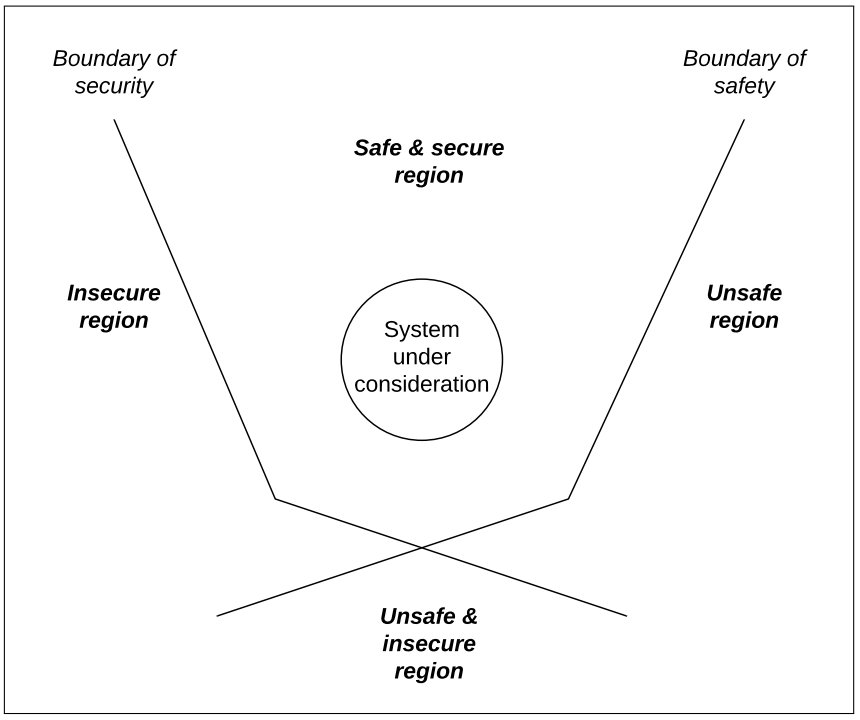

Figure 7.1: Representation of the "boundary of acceptable performance" concept as applied to safety and security jointly

This model serves as the basis for a methodology which seeks to analyse both security and safety as first-class citizens of the analysis. Previous contributions which involved expanding STPA to apply to security, as detailed in Section 3.4, do not provide a conceptual model to frame the approach, instead only seeking to modify the analysis technique to include security analysis aspects. This deprives the new methodology of a firm theoretical basis for interpreting and understanding the inter-relation between safety and security behaviours of a system.

Considering system actors and mitigations within the theoretical model
Another contribution of this theoretical model is that it provides a more rigorous basis for understanding the behaviours of all individuals involved with the system when

compared to Leveson's safety-focused model. In the STAMP model, the understanding of the primary influences on the safety of the system was through the consideration of *authorised users* who may take actions which unintentionally degrade the safety of the system. There were many stated reasons why authorised users may take these actions such as optimisation of work patterns, a desire to reduce costs, etc. but the clear view was these were authorised users who needed to be made aware consistently throughout the system life-cycle of how to maintain the system in a safe state. This was best achieved through placing *constraints* on the system behaviours, or on the operators/controllers associated with the system, to ensure that hazardous states could not be entered into. The notion of unwanted or *malicious* actors who may be actively seeking to undermine the system is not a primary consideration within the theoretical framework, although the STPA technique does feature some consideration of security as it relates to safety.

The STAAMP theoretical model enables a more thorough consideration, when compared with STAMP, of all individuals involved in the system as it can conceptualise the processes behind a system approaching the boundary of safety and security as significantly different in origin and motivations. While it reuses the understanding from STAMP that the maintenance of safety is primarily focused on curtailing unsafe or hazardous actions and decisions from operators and system controllers, it conceptualises the process of maintaining system security as a process of curtailing the actions of *motivated, unauthorised actors* in taking *intentional* actions which degrade the security of the system. There is additionally consideration of ordinarily-authorised users taking unintentional actions which may degrade the system security, as this is another relevant factor in maintaining system security. In either case, the focus of any mitigation of malicious actor activity must be an aspect of the system design; it is not possible to constrain the *behaviours* of malicious actors as they are, by definition, acting outside the usual rules and procedures that govern the correct (i.e. safe and secure) operation of the system in question. A minority of constraints may be placed on authorised users to ensure they do not unintentionally reduce the security of the system (such as enforcing unique passwords) but these are essentially secondary concerns in compared to the primary means through which system security is degraded. A high-level comparison of the approaches of STAMP and STAAMP can be found in Table 7.1.

Table 7.1: Comparison of approaches to security and safety

Model	Approach to security	Approach to safety
STAMP	Minor consideration of security in relation to safety	Primary focus; safety understood as a failure of control by either system controllers or operators within the system; mitigations placed on system design and operators equally.
STAAMP	Security understood in terms of intentional actions taken by malicious users; mitigations must be built into system design as the majority of malicious users cannot be constrained directly.	Safety understood as a failure of control by either system controllers or operators within the system; mitigations placed on system design and operators equally.

7.2.2.2 Contribution of the SE-STPA technique

Once the theoretical model was in place to contextualise the relationship between safety and security, the SE-STPA methodology was created. This methodology provides a distinct step for security analysis step which was termed 'adversarial modelling'. This step involves the identification of an *adversary* who seeks to undermine the system's security in some manner. The construction of the adversary during this step of the analysis seeks to provide an explicit capture of possible threat actors against the system, as well as their capabilities and intent. The purpose of this step is to enable consideration of how existing system behaviours and design features can either permit or deny the attacker from undermining the system. In this regard, it acts as an explicit security assurance case, to demonstrate that the design adequately handles attacks from threat actors identified during the adversarial modelling step of analysis. It also provides a traceable artefact to which *critical requirements* can be attached. Further detail on the adversary modelling step can be found in subsubsection 4.2.4.6 and a summary is provided here for brevity.

Furthermore, in order to enable this analysis to be undertaken in a unified way, the terminology for both security and safety steps was unified. This was a decision which enabled a unified lexicon to be used throughout the process of applying SE-STPA to a system. This was done in order to ensure that confusion would not occur between analysts when performing the safety analysis steps and the security analysis steps. The

decisions made in regards to unifying the terminology, and their justifications, can be found in subsection 3.7.4.

Finally, SE-STPA provides two additional points at which security hazards can determined. These two points primarily focus on individual control actions and how these can become hazardous from both a safety and security viewpoint, and therefore serves as a complementary step to adversarial modelling which focuses on a broader notion of a threat actor in the form of an adversary, who can take a number of actions.

The first point of additional security analysis is during the control action analysis step subsubsection 4.2.4.5 where the generic terminology of 'hazard' is utilised in the control action analysis step. Due to the generic terminology, this step enables safety issues to be determined (as with standard STPA) but it also enables the identification of security issues relating to an individual control action. This is similar in some regards to the approach taken by STPA-Sec, but STPA-Sec utilises the terminology of 'insecure or unsafe' which results in the reader context-switching between safety and security domains, rather than considering it as a unified whole.

The second point of additional security analysis is during the step known either as 'scenario generation' or 'causal factors analysis' in baseline STPA, where control actions are subjected to analysis for *how* they can become hazardous[1]. This enables issues such as human factors to be considered in the baseline STPA methodology, but once again, the unified terminology of *hazard* in SE-STPA relating to both security and safety enables thoughtfulness about causal factors that can contribute to hazardous control actions emerging.

A copy of Figure 4.1 with an emphasis on steps relevant to security analysis can be found in Figure 7.2.

[1]SE-STPA maintains the naming of this step as 'causal factors analysis' as the methodology was substantially developed prior to the creation of the current version of the STPA Handbook, which re-titles this step.

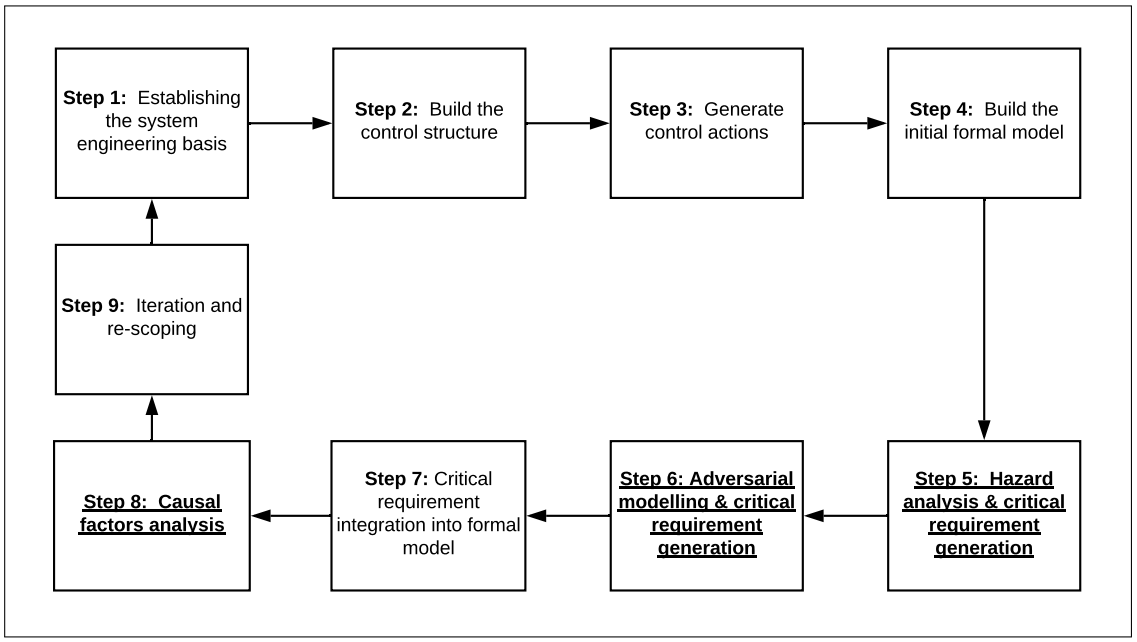

Figure 7.2: Steps of SE-STPA with an emphasis on security-relevant steps

When contrasted with existing approaches to enabling security analysis using STPA, SE-STPA therefore has some distinct advantages:

1. No existing co-analysis techniques, whether based on STPA or standalone methodologies such as CHASSIS, provide a detailed assurance case of which threat actors are explicitly considered during security analysis. The identification of a threat actor in the form of an adversary enables a clear, traceable understanding of what a given threat actor is perceived to be able to do the system in terms of *actions* and what they are perceived to be able to *understand* in terms of system information. SE-STPA provides all of this in the adversarial modelling stage.

2. Standalone security analysis techniques do not easily integrate with safety analysis techniques due to the conflicting terminology often used by the two separate domains. The unified terminology utilised in SE-STPA ensures that whatever the source of a hazard (i.e. whether it be from security or safety analysis), there will always be a *critical requirement* generated to mitigate it. The set of critical requirements generated during analysis can then be considered as a whole to ensure there are not contradictory critical requirements between safety and security analysis steps.

3. This approach maintains the broad applicability of STPA across the entire system life-cycle, while also providing a number of opportunities and manners for security issues (in the form of hazards) to be identified and mitigated. This in direct contrast to a number of security analysis techniques which are focused on implementation-specifics or design-specifics, and so cannot be applied outside

of their intended life-cycle phase. This is additionally true for some co-analysis methodologies, such as STPA-SafeSec [51], which involve security analysis techniques which require an implementation of the system to have been undertaken to actually use the techniques.

7.2.3 Limitations

The limitations of the contribution to this research question at present can be broadly described as follows:

1. It is not clear whether the methodology would scale successfully to large cyber-physical systems. In the absence of a large case study to determine this one way or another, this must be declared as a potential limitation. Nonetheless, the methodology scaled well from the smaller smart meter case study to the more-complex multi-UAV case study.
2. The 'adversarial modelling' analysis step is relatively immature in terms of its usage by a number of practitioners on a number of case studies and systems when compared to the baseline aspects of STPA, and therefore a number of future validation activities may demonstrate that there are more effective means of undertaking co-analysis than the use of this concept.

7.2.4 Summary

The major contributions as presented in this section towards addressing this research question are two-fold:

1. The systems-theoretic model in the form of STAAMP which conceptualises security and safety concepts for cyber-physical systems in systems theory terms, and serves a conceptual foundation for a co-analysis methodology to be developed.
2. The systems-theoretic analysis technique in the form of SE-STPA which includes security as a first-class citizen of the analysis through unified terminology, a robust security analysis step that is broadly applicable throughout the system life-cycle and a number of other steps which enable the identification of both security and safety hazards in relation to maintaining correct control of the system.

The research question can therefore be considered to have been met by these contributions.

7.3 Discussion of Research Question 2

Can the methodology additionally leverage formal method techniques in order to provide assurance that the mitigations generated as a result of analysis are demonstrably successful in reducing or eliminating security and safety hazards?

7.3.1 Context and approach

In the baseline STPA methodology, constraints are generated to address unsafe control actions, and this analysis is intended to be undertaken with expert review to inform the generation of the constraints. However, when considering safety and security jointly as intended by Research Question 1 and embodied by SE-STPA , it is entirely possible and likely that a number of critical requirements will conflict or require contradictory system behaviours. This may not be picked up until a subsequent analysis loop, or potentially even during implementation, and so ensuring that the critical requirements as a set are consistent and non-contradictory before refining the design is important to ensuring confidence in the methodology as a whole.

In response to this, a formal method technique was to be integrated into SE-STPA in order to provide further assurance or rigour that critical requirements were consistent and enabled the system to still maintain its purpose. The formal method chosen was Event-B; a summary of the formal method is provided in subsubsection 3.7.3.1 while the benefits of this formal method are provided in subsubsection 3.7.3.2.

7.3.2 Contribution

The integration of the Event-B formal method into the methodology is primarily undertaken in two major steps of the analysis:

- In Step 4 of the analysis, once the system's control structure and actions are understood, as detailed in subsubsection 4.2.4.4.
- In Step 7 of the analysis, the set of critical requirements generated in prior steps is integrated into the formal model, as detailed in subsubsection 4.2.4.7.

A copy of Figure 4.1 with an emphasis on steps relevant to the involvement of the formal method can be found in Figure 7.3. Examples of how the formal method was used for translating critical requirements into constraints on the system model within each case study can be found in subsubsection 5.3.6.1 for the smart meter case study and subsubsection 6.2.7.1 for the multi-UAV case study.

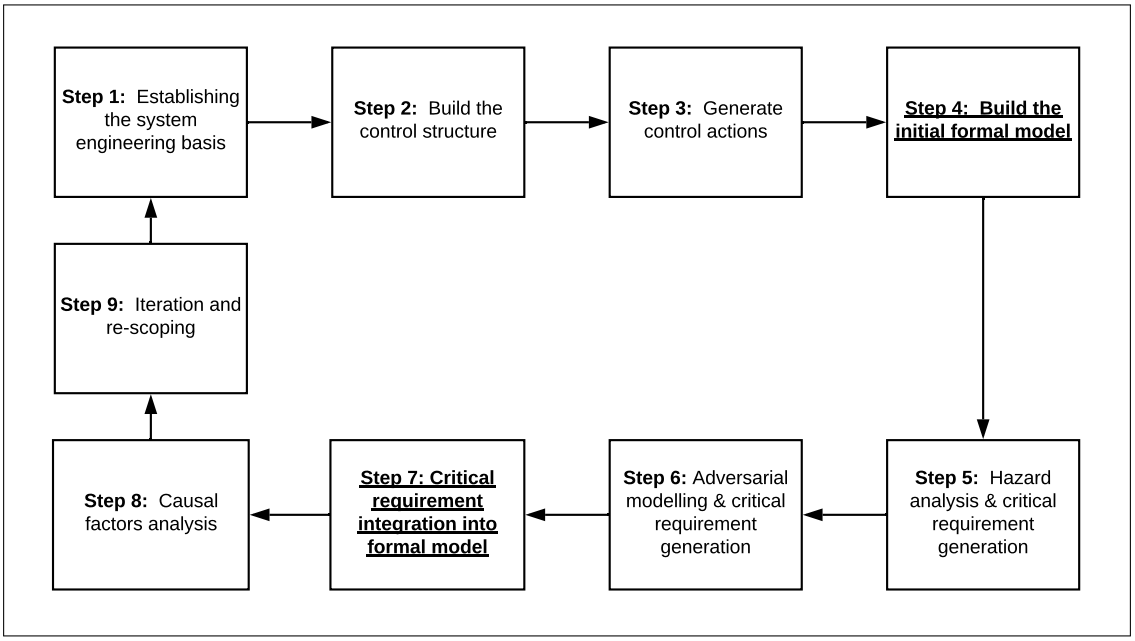

Figure 7.3: Steps of SE-STPA with an emphasis on formal method-relevant steps

The contribution of the formal method in assuring that safety and security critical requirements are consistent can be broadly summarised as follows:

1. The construction of the formal method prior to undertaking any detailed analysis forces the analyst to consider whether the system's behaviours are sufficiently defined in order to model them in Event-B terms. This increases thoughtfulness around the system specification and documentation, and enables clarification of the system behaviours prior to the analysis truly beginning.

2. The integration of the critical requirements into the formal model enables a number of features which can be used for validating that the critical requirements as a whole are consistent and do not either introduce contradictory or otherwise undesirable behaviours into the system:

   (a) Contradictory critical requirements will result in the model consistently violating invariants when animated, as critical requirements are primarily modelled as invariants when they restrain system behaviours. A trace of all events (i.e. state transitions) can be reviewed to indicate which invariants are likely contradictory, and this can enable modification or unification of the critical requirements into a single, non-contradictory critical requirement.

   (b) Critical requirements that impose contradictory limits on variables will be flagged by the tool before any animation is possible, as variable types must be consistent.

(c) Critical requirements modelled as invariants and guards may result in the system entering into deadlock states where no further transitions are enabled. This enables these critical requirements to be taken away for further analysis and reworked to ensure the system model can behave as required, while still respecting the critical requirements[2].

(d) The use of animation also enables the model execution to be simulated for a time and the state of each variable to be viewed at each step. This is used to provide a higher degree of confidence[3] that states that critical requirements should prevent (i.e. a forbidden state occurs despite the associated critical requirement being modelled) are never entered by the formal model.

3. The generation of critical requirements becomes more specific as the analyst is aware that they will need to be entered into a formal model at a later point in the analysis. This can focus attention and ensure that nebulous critical requirements are not generated, as the level of specificity required to model them is also likely to be required during implementation of the critical requirements on the real system.

4. The generation and discharging of proof obligations provides a formal proofs of system behaviours. This is helpful in ensuring that the system model behaves in a well-defined way, as there will be formal proofs to demonstrate that even with a number of critical requirements integrated into the system model, it is possible to discharge all proof obligations.

5. The formal model provides an additional 'hook' in traceability terms; each refinement to the formal model embodies a critical requirement, which mitigates against a hazard, which connects back up to the system engineering context of the system such as Purposes, Losses and system-level Hazards. This enables a clear, traceable link to be demonstrated when writing assurance cases against the system, as is likely to be required in highly regulated domains and industries.

The use of the Event-B formal method provides a degree of assurance of critical requirements that is more substantial than that provided by the baseline STPA methodology, which is highly reliant on expert review and iterative analysis to identify flaws in generated constraints. The use of the formal model with a co-analysis methodology is a novel contribution within the literature and serves to reduce the burden of, as well as relieve some of the dependency on, domain experts undertaking analysis of cyber-physical systems. This is particularly important when one considers the ever-increasing use of cyber-physical systems in a number of domains. Additionally, it can serve to reduce the

[2]Deadlock state checking as provided by the Rodin tool can execute for an unbounded amount of time and still not guarantee that deadlock states are not entered into. Nonetheless, it provides a degree of confidence of correctness that is not otherwise possible were the formal model not used.

[3]Model animation possesses an unbounded state space for anything beyond the most trivial model, and so this can only provide a higher degree of confidence, not absolute certainty

cost and expense involved in revising the system design or implementation to integrate the results of the analysis in the form of critical requirements, as the critical requirements will essentially have been 'vetted' by integrated into the formal method prior to design modifications taking place.

7.3.3 Limitations

The primary limitations of the Event-B formal method are common to a number of formal method techniques, but bear repeating for the purpose of this thesis.

7.3.3.1 Expertise in formal methods

The use of the formal method technique as a core aspect of the analysis relies on there being expertise within the analysis team in the use of formal methods techniques, and specifically one who understands the syntax and structure of Event-B formal models, as well as the associated tools. A lack of this expertise may result in incomplete or poorly defined formal models, which do not serve as a strong basis for validating the correctness of critical requirements. However, in the worst-case scenario where the formal method is discarded and the formal method is not utilised during analysis, the resulting analysis is no worse than a standard, baseline application of STPA to the system in terms of validating the correctness of critical requirements.

7.3.3.2 Representing complex critical requirements

The representation of particularly advanced critical requirements may be challenging within the Event-B formal method environment. As the system design becomes more concrete and specific, security critical requirements may hinge on the correct use of asymmentric encryption as an example, or by a determinism that is difficult to model within Event-B. Once again however, dispensing with the formal method results in a similar situation to baseline STPA application in terms of a lack of formal validation of critical requirements. Furthermore, the Rodin tool for Event-B has a number of plugins - such as the theory plug-in [24], the decomposition plug-in [130] and the qualitative probability plug-in [57] - which may enable these concepts to be modelled sufficiently.

7.3.4 Summary

The major contributions presented in this section demonstrate that there is significant value in the use of the Event-B formal method for validating the critical requirements generated by other steps of the methodology. Event-B and the associated tooling enables

a degree of thoughtfulness and analysis that would not otherwise be possible if the critical requirements were immediately integrated into the system design. While the Event-B approach and associated tooling have some limitations relevant to this thesis, it is still a significant improvement above the baseline STPA technique. The use of a formal method to support a co-analysis approach in this way is also a novel contribution of this thesis as prior attempts in the literature have not used the formal method in such a heavily-integrated way. **The research question can therefore be considered to be met.**

7.4 Discussion of Research Question 3

Can the methodology demonstrate its utility through application to case studies of differing sizes and complexity, ideally utilising case studies involving cyber-physical systems in multiple domains?

7.4.1 Context and approach

The approach to this research question was to apply the methodology at varying points of its development to case studies which were sourced from public information or the existing literature. These were selected from two separate domains to ensure the methodology was broadly applicable to all cyber-physical systems, and not just those belonging to a single domain.

The intent of applying the methodology to case studies was to undertake a 'lessons learned' review at the end of each case study, and to use the results to inform improvements to the methodology.

7.4.2 Contribution

The two case studies selected were as follows:

1. A case study based on a simplified smart meter system, as detailed in Chapter 5.
2. A case study based on system comprising of a number of unmanned aerial vehicles (multi-UAV), as detailed in Chapter 6.

As each of these case studies were undertaken at different maturities of SE-STPA , their contributions are broken down in the following sections.

7.4.2.1 Smart meter case study

This case study was based around a simplified version of a smart metering system for electricity, based on a proposed national architecture for smart metering technologies. It was therefore 'synthetic' as it was generated from a number of public information sources and was felt to have been the appropriate size and scale in applying the second version of SE-STPA .

This case study involved the application of the second version of the methodology and so was primarily beneficial in developing the methodology into the third version described in Chapter 4. This case study revealed some flaws with the second iteration of the methodology which drove the following improvements[4]:

- Security analysis was made more prominent and the adversary modelling step was added, which represents an explicit security assurance case for a number of relevant threat actors.
- Critical requirements are primarily captured in two steps, Control Action Analysis and Adversary Modelling, and then are integrated into the formal model rather than being done in an ad-hoc or unstructured manner.
- The methodology was better defined and so was clearer and easier to apply to case studies.

The primary contribution of this case study was therefore in providing structural improvements in how SE-STPA is undertaken as well as providing an opportunity for the first medium-scale application of the methodology. A secondary contribution was that it also served to highlight that the core concept of the methodology, including the formal method, was sound as there were no fundamental issues in applying it to the smart meter case study. It was therefore decided that a more substantial case study could be utilised once the methodology had been iterated on.

7.4.2.2 Multi-UAV case study

This case study served as a more substantial test of the methodology. It was based on an existing multi-UAV system as described in [19] and therefore involved a number of entities which were cyber-physical in nature. There was also a deeper complexity to each of the entities, as they maintained distinct process models and information of their own.

[4]Further detail on the identified issues and resulting improvements can be found in Section 5.4

The primary contribution of this case study was to provide a significant test of the methodology in its third version. This involved the following novel aspects compared to prior case studies:

1. The first extensive use of the adversarial modelling concept and analysis step. This had been used in an ad-hoc manner with some small, informal case studies but had not been used as part of the integrated methodology until this case study.
2. The use of the formal method in a more disciplined, structured way. This involved the creation of an Event-B model that completely represented the defined functionality of the system prior to the safety & security analysis occurring, whereas previous attempts had used the formal method in a less prescriptive manner.
3. The approach of undertaking the bulk of the analysis and generating the critical requirements, before seeking to integrate all generated critical requirements into the formal model, ensured that the bulk of critical requirements (exempting those from causal factors) would be validated in the formal model before the system design was iterated.
4. The size of the case study was more substantial in terms of entities and analysis required than the preceding one, and so involved a more thorough application of the methodology than had previously been achieved.

The multi-UAV case study served as a useful validation step in the development of the methodology, and ensured that the current version of the methodology was sound for future case studies or other validation activities.

7.4.3 Limitations

A limitation of the case studies is that only two were achieved during the course of the thesis. This was primarily driven by the lack of public information regarding the detailed behaviours of cyber-physical systems in key domains and so the construction of case studies was laborious; often it was not possible to develop a case study of the appropriate scope and size to serve as a suitable test of the methodology and its approach. Nonetheless, the two case studies undertaken in validating the methodology were effective in proving the core concept of the methodology, and provided an opportunity to iteratively improve the methodology both conceptually and in terms of the precise steps of analysis that were performed. Furthermore, the first case study enabled the author to identify that a much more substantial and detailed approach to security analysis was required, and this drove the improvement of the third version of SE-STPA which resulted in security having a significant role in the overall analysis.

A secondary limitation is that the use of only two case studies from the infrastructure and transport domains may have cyber-physical challenges that are similar, while the selection of a third case study from another domain using cyber-physical systems (e.g. maritime) may have provided a number of additional challenges not present in either case study. This is a driver for future work to contain further validation activities, including case studies from other industries, and is an unavoidable limitation at this stage. Work by Omitola et al. [105] may represent an opportunity for future collaboration in applying SE-STPA to cyber-physical systems in the marine domain.

7.4.4 Summary

In summary, the methodology was applied in two case studies involving cyber-physical systems in two separate domains. The application of these case studies enabled the core concept of the analysis to be validated, and provided an opportunity to increase the scale of the analysis as the methodology matured. While further case studies would have likely benefited in refining the methodology further, **the research question can be considered to have been met** as SE-STPA has been applied to two case studies and demonstrated utility in both.

7.5 Additional academic review of the methodology

A further activity during the construction and improvement of the methodology was the authoring of two papers; one for a conference workshop, later published in its proceedings, and one in a peer-reviewed journal. This activity sought to ensure that the methodology had been through some degree of academic peer review as part of its development.

The first paper involved a presentation of the methodology, against a low-complexity example, at the 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW); the specific workshop was the 2nd International Workshop on Safety & Security aSSurance for Critical Infrastructures Protection [62]. This paper represented the second version of SE-STPA and therefore had not benefited from the lessons learned from the smart meter case study.

The second paper was published in the International Journal of Critical Computer-Based Systems [61] and represented a methodology that was broadly equivalent to the third version of SE-STPA . This paper presented the adversarial modelling concept for the first time, as well as applying this against the smart meter case study. The peer review process for this paper informed a number of smaller changes to the methodology, which are embodied in the version of SE-STPA presented in Chapter 4.

Chapter 8

Conclusion

This thesis proposes a theoretical model in the form of STAAMP and associated analysis methodology in the form of SE-STPA for undertaking security and safety co-analysis. It does this by building on Leveson's work in STPA and STAMP to create a methodology which treats security and safety as first-class citizens of the analysis. Additionally, robustness is provided through the use of the Event-B formal method, which has been integrated to distinct steps of the analysis to ensure that all mitigations produced to address hazards are validated in the formal model before they are integrated into the design.

This analysis methodology has been validated through application to two case studies involving cyber-physical systems from different domains, and has demonstrated itself as being adept and thorough in its ability to identify both security and safety hazards and propose mitigations which have been formally validated. The thesis has therefore addressed the three research questions as detailed in Chapter 7.

8.1 Contributions

The contributions of this thesis can be summarised as follows:

1. A theoretical model, STAAMP, which provides a conceptual framework for understanding the inter-relation of security and safety from a systems-theoretic perspective. While this is used within this thesis as the basis for the SE-STPA methodology, it may also be utilised in understanding security and safety in other contexts, such as when performing analysis of system failures.

2. An analysis methodology, SE-STPA, which is capable of performing security & safety analysis jointly on cyber-physical systems in order to provide assurance that safety and security hazards have been identified, mitigated in the form of

critical requirements, and the underlying design corrected for these systems. The use of a formal model throughout this process also provides a higher degree of assurance that critical requirements do indeed mitigate the hazards they seek to address.

3. Two case studies which may be useful for testing other co-analysis methodologies in a cyber-physical systems context; one of low complexity and one of medium complexity.

4. A literature review, covering a number of existent techniques in both security analysis, safety analysis and co-analysis. Furthermore, the literature review contains a high-level review of formal method techniques as applied to the safety, security and co-analysis domains. This survey can be utilised as a starting point for other cyber-physical researchers with an interest in these topics.

5. An exploration of the STPA family of methodologies as well as perceived shortcomings in terms of general shortcomings and those specific to the security domain.

8.2 Future work

A number of avenues of future work can be foreseen in terms of both iterative improvements to the methodology and additional steps which could be utilised in pursuit of validating the methodology further.

8.2.1 Iterative improvements to the methodology

Modelling of trust between entities As previously explored in Section 6.3, the current methodology does not model trust in a significant way, although this is often an aspect of system design when considering cyber-physical systems. Both safety and security can result in varying degrees of trust of components involved in the cyber-physical system; an example of this from a safety perspective would be voting between sensor channels to ensure there is tolerance in the event of a single sensor failure, while a security example of trust behaviours would be treating all data from a physically-remote system which may be tampered with as suspect and subject to further analysis/validation before accepting the value. A future version of SE-STPA could incorporate a trust model as part of the analysis, which may aid the process of carrying out co-analysis.

Improvements in formal traceability In order to utilise the methodology in highly regulated domains, it is important that the methodology generates uniquely identified artefacts which can form part of a traceability chain. This would permit the artefacts of the analysis to be utilised in assurance cases and other activities as required in these

domains. The process for improving this would require the generation of standard templates for carrying out the analysis, as well as explicitly requiring each step of the analysis to map artefacts it generates to those generated in prior stages of the analysis.

Tool support The current methodology utilises Rodin as the tool support for the use of the formal method, but does not use any other tools that are specific to the methodology. A number of tools exist for performing STPA analysis or managing artefacts generated as part of the analysis [1, 141] and so future work could explore the creation of tools to streamline the use of SE-STPA . As Rodin is built on the extensible Eclipse framework [2], this may be an avenue of exploration.

8.2.2 Additional validation steps

Workshop with experienced practitioners Validation of the methodology would significantly benefit from a workshop with domain experts on security & safety of cyber-physical systems, where the methodology is applied to a smaller-scale case study, and feedback from the attendees on the strengths and weaknesses of the methodology from their experience. Such a validation activity would require identification of a suitable group of domain experts, as well as an appropriate case study, as well as ethical approval due to human involvement. Nonetheless, this would be a useful next step in gaining further feedback on the methodology and iterating on it further.

Industrial case studies A further useful validation activity would be to take SE-STPA and apply it to a cyber-physical system in an industrial context. This would involve working with local subject matter experts and would provide a degree of 'real world' validation that many of the techniques in Chapter 2 have benefited from.

Another example of an industrial case study approach which would further validate and provide improvement to SE-STPA would be through comparison of the existing two case studies provided in Chapter 6 and Chapter 5 to other analyses undertaken against this class of cyber-physical system. For the smart meter case study given in Chapter 5, it could be compared with work undertaken by the National Cyber Security Centre in assessing the national smart meter infrastructure [93] and the results used to inform improvements in SE-STPA.

Direct comparison with other co-analysis methodologies A final useful avenue would be to undertake a SE-STPA analysis of a system or case study for which another co-analysis technique had been applied. This analysis would be applied 'blind', with no knowledge of the results from the other co-analysis methodology, and then the results compared to determine the comparative coverage of SE-STPA.

Comparison of adversary modelling to existing security best practice frameworks A further validation activity would involve the comparison of adversary modelling aspect of SE-STPAspecifically to existing security best practice frameworks for cyber-physical systems and/or Internet of Things (IoT) devices. This work would draw upon existing frameworks such of that proposed by the Internet of Things Security Foundation (IoTSF) [64] and/or the European Union Agency for Cybersecurity (ENISA) [48] and would provide a demonstration of how effective the adversary modelling concept is in drawing out similar issues and mitigations when compared to these frameworks.

Appendix A

Smart meter case study - initial formal model - machine & context

CONTEXT SmartMeterContext

SETS

METERS

BILLS

READINGS

END

MACHINE SmartMeterMachine

SEES SmartMeterContext

VARIABLES

RegisteredMeters

RetiredMeters

MeterReading

BillGenerated

BillPaid

CurrentWeek

AllBills

MeterBill

ReadingValue

ReadingReported

AllReadings

MeterDisconnectState

MeterRegisteredDate

INVARIANTS

RegisteredMeterTypeInvariant: $RegisteredMeters \subseteq METERS$

RetiredMeterTypeInvariant: $RetiredMeters \subseteq METERS$

MeterReadingTypeInvariant: $MeterReading \in METERS \leftrightarrow READINGS$

AllBillsTypeInvariant: $AllBills \subseteq BILLS$

BillGeneratedTypeInvariant: $BillGenerated \in AllBills \pfun \mathbb{N}$

BillPaidTypeInvariant: $BillPaid \in BILLS \leftrightarrow BOOL$

CurrentWeekTypeInvariant: $CurrentWeek \in \mathbb{N}$

MeterBillTypeInvariant: $MeterBill \in RegisteredMeters \cup RetiredMeters \leftrightarrow\!\!\!\!\!\rightarrow AllBills$

AllReadingsTypeInvariant: $AllReadings \subseteq READINGS$

ReadingValueTypeInvariant: $ReadingValue \in AllReadings \leftrightarrow \mathbb{N}$

ReadingReportedTypeInvariant: $ReadingReported \in AllReadings \leftrightarrow \mathbb{N}$

MeterDisconnectStateTypeInvariant: $MeterDisconnectState \in METERS \pfun BOOL$

MeterRegisteredDateTypeInvariant: $MeterRegisteredDate \in RegisteredMeters \pfun \mathbb{N}$

EVENTS

Initialisation

begin

AllBillsInit: $AllBills := \varnothing$

RegisteredMeterInit: $RegisteredMeters := \varnothing$

RetiredMetersInit: $RetiredMeters := \varnothing$

MeterReadingsInit: $MeterReading := \varnothing$

CurrentWeekInit: $CurrentWeek := 0$

BillGeneratedInit: $BillGenerated := \varnothing$

BillPaidInit: $BillPaid := \varnothing$

AllReadingsInit: $AllReadings := \varnothing$

MeterDisconnectStateInit: $MeterDisconnectState := \varnothing$

ReadingValueInit: $ReadingValue := \varnothing$

ReadingReportedInit: $ReadingReported := \varnothing$

MeterBillInit: $MeterBill := \varnothing$

MeterRegisteredDateInit: $MeterRegisteredDate := \varnothing$

end

Event RegisterMeter ⟨ordinary⟩ $\widehat{=}$

any

meter

where

meterTypeGuard: $meter \in METERS$

then

RegisterMeterAction: $RegisteredMeters := RegisteredMeters \cup \{meter\}$

MeterSetRegistrationDateAction: $MeterRegisteredDate(meter) := CurrentWeek$

MeterDisconnectedInit: $MeterDisconnectState(meter) := FALSE$

end

Event RetireMeter ⟨ordinary⟩ $\widehat{=}$

any

meter

where

meterTypeGuard: $meter \in RegisteredMeters$

then

RetireMeterAction: $RetiredMeters := RetiredMeters \cup \{meter\}$

RemoveRegisteredMeter: $RegisteredMeters := RegisteredMeters \setminus \{meter\}$

RemoveMeterRegDate: $MeterRegisteredDate := \{meter\} \lhd MeterRegisteredDate$

RemoveDisconnectState: $MeterDisconnectState := \{meter\} \lhd MeterDisconnectState$

end

Event GenerateBill ⟨ordinary⟩ $\widehat{=}$

any

meter

bill

where

meterTypeGuard: $meter \in RegisteredMeters$

billTypeGuard: $bill \in BILLS$

currentWeekType: $CurrentWeek \in \mathbb{N}$

then

AddBillAction: $AllBills := AllBills \cup \{bill\}$

MarkGeneratedBillAction: $BillGenerated(bill) := CurrentWeek$

MarkBillUnpaidAction: $BillPaid(bill) := FALSE$

MeterBillAssign: $MeterBill := MeterBill \cup \{meter \mapsto bill\}$

end

Event PayBill ⟨ordinary⟩ $\widehat{=}$

any

meter

bill

where

meterTypeGuard: $meter \in RegisteredMeters$

billTypeGuard: $bill \in AllBills$

then

setBillPaidAction: $BillPaid(bill) := TRUE$

end

Event AddMeterReading ⟨ordinary⟩ $\widehat{=}$

any

meter

reading

readingValue

where

meterTypeGuard: $meter \in RegisteredMeters$

readingTypeGuard: $reading \in READINGS$

currentWeekType: $CurrentWeek \in \mathbb{N}$

readingValueType: $readingValue \in \mathbb{N}$

then

AllReadingsAddAction: $AllReadings := AllReadings \cup \{reading\}$

AddMeterReadingAction: $MeterReading := MeterReading \cup \{meter \mapsto reading\}$

ReadingValueAction: $ReadingValue(reading) := readingValue$

ReadingReportedAction: $ReadingReported(reading) := CurrentWeek$

end

Event AdvanceTime ⟨ordinary⟩ $\widehat{=}$

begin

CurrentWeekAdvance: $CurrentWeek := CurrentWeek + 1$

end

Event DisconnectMeter ⟨ordinary⟩ $\widehat{=}$

any

meter

where

meterTypeGuard: $meter \in RegisteredMeters \land meter \in dom(MeterRegisteredDate)$

disconnectCriteria: $(\forall x, y, z \cdot x = MeterBill[\{meter\}] \land y = (x \lhd BillGenerated) \rhd \{item | item \in CurrentWeek - 999 \,..\, CurrentWeek - 8\} \land z = (dom(y) \lhd BillPaid) \Rightarrow (FALSE \in ran(z)) \land (MeterRegisteredDate(meter) + 12 \leq CurrentWeek)) \lor (\forall x, y \cdot x = MeterReading[\{meter\}] \land y = (x \lhd ReadingReported) \rhd \{item | item \in CurrentWeek - 4 \,..\, CurrentWeek - 1\} \Rightarrow y = \varnothing \land MeterRegisteredDate(meter) + 12 \leq CurrentWeek)$

then

DisconnectMeterAction: $MeterDisconnectState(meter) := TRUE$

end

END

Appendix B

Smart meter case study - final formal model - machine & context

CONTEXT SmartMeterContextFull

EXTENDS SmartMeterContext

SETS

TOKENS Added to address CR.5.

END

MACHINE SmartMeterMachineRefinement

REFINES SmartMeterMachine

SEES SmartMeterContextFull

VARIABLES

RegisteredMeters

RetiredMeters

MeterReading

BillGenerated

BillPaid

CurrentWeek

AllBills

MeterBill

ReadingValue

ReadingReported

AllReadings

MeterDisconnectState

MeterRegisteredDate

AllTokens CR5

RegisteredTokens CR5

INVARIANTS

`AllTokensTypeInvariant`: $AllTokens \subseteq TOKENS$

CR5

`RegisteredTokensTypeInvariant`: $RegisteredTokens \in RegisteredMeters \rightarrowtail AllTokens$

CR5

`MetersRetireOverlapInvariant`: $RegisteredMeters \cap RetiredMeters = \varnothing$

CR1

EVENTS

Initialisation ⟨extended⟩

begin

`AllBillsInit`: $AllBills := \varnothing$

`RegisteredMeterInit`: $RegisteredMeters := \varnothing$

`RetiredMetersInit`: $RetiredMeters := \varnothing$

`MeterReadingsInit`: $MeterReading := \varnothing$

`CurrentWeekInit`: $CurrentWeek := 0$

`BillGeneratedInit`: $BillGenerated := \varnothing$

`BillPaidInit`: $BillPaid := \varnothing$

`AllReadingsInit`: $AllReadings := \varnothing$

MeterDisconnectStateInit: $MeterDisconnectState := \varnothing$

ReadingValueInit: $ReadingValue := \varnothing$

ReadingReportedInit: $ReadingReported := \varnothing$

MeterBillInit: $MeterBill := \varnothing$

MeterRegisteredDateInit: $MeterRegisteredDate := \varnothing$

AllTokensInit: $AllTokens := \varnothing$

RegisteredTokensInit: $RegisteredTokens := \varnothing$

end

Event RegisterMeter ⟨ordinary⟩ $\widehat{=}$

extends RegisterMeter

any

$meter$

token

where

meterTypeGuard: $meter \in METERS$

NotAlreadyRetiredGuard: $meter \notin RetiredMeters$

CR1

tokenTypeGuard: $token \in TOKENS$

CR5

NotAlreadyRegisteredGuard: $meter \notin RegisteredMeters$

CR2

tokenUniquenessUnassignedGuard: $token \notin ran(RegisteredTokens)$

CR5

tokenUniquenessAbsentGuard: $token \notin AllTokens$

CR5

then

RegisterMeterAction: $RegisteredMeters := RegisteredMeters \cup \{meter\}$

MeterSetRegistrationDateAction: $MeterRegisteredDate(meter) := CurrentWeek$

MeterDisconnectedInit: $MeterDisconnectState(meter) := FALSE$

tokenAssignAction: $RegisteredTokens(meter) := token$

CR5

tokenAddAction: $AllTokens := AllTokens \cup \{token\}$

CR5

end

Event RetireMeter ⟨ordinary⟩ $\widehat{=}$

extends RetireMeter

any

$meter$

where

meterTypeGuard: $meter \in RegisteredMeters$

RetireMeterMayNotHaveBillGuard: $\forall x, y \cdot x = MeterBill[\{meter\}] \land y = x \lhd BillPaid \Rightarrow FALSE \notin ran(y)$

CR3

then

RetireMeterAction: $RetiredMeters := RetiredMeters \cup \{meter\}$

RemoveRegisteredMeter: $RegisteredMeters := RegisteredMeters \setminus \{meter\}$

RemoveMeterRegDate: $MeterRegisteredDate := \{meter\} \lhd MeterRegisteredDate$

RemoveDisconnectState: $MeterDisconnectState := \{meter\} \lhd MeterDisconnectState$

removeTokenAction: $RegisteredTokens := \{meter\} \lhd RegisteredTokens$

CR5

end

Event GenerateBill ⟨ordinary⟩ $\widehat{=}$

extends GenerateBill

any

$meter$

$bill$

where

meterTypeGuard: $meter \in RegisteredMeters$

billTypeGuard: $bill \in BILLS$

currentWeekType: $CurrentWeek \in \mathbb{N}$

billUniqueGuard: $bill \notin AllBills$

CR7

CurrentWeekGuard: $CurrentWeek mod 4 = 0$

CR9

NoDoubleBillGuard:

$\forall checkBill, temp \cdot checkBill = MeterBill[\{meter\}] \land temp \in checkBill \Rightarrow temp \in dom(BillGenerated) \land CurrentWeek \neq BillGenerated(temp)$

CR9

then

AddBillAction: $AllBills := AllBills \cup \{bill\}$

MarkGeneratedBillAction: $BillGenerated(bill) := CurrentWeek$

MarkBillUnpaidAction: $BillPaid(bill) := FALSE$

MeterBillAssign: $MeterBill := MeterBill \cup \{meter \mapsto bill\}$

end

Event PayBill ⟨ordinary⟩ $\widehat{=}$

extends PayBill

any

$meter$

$bill$

token

where

meterTypeGuard: $meter \in RegisteredMeters$

billTypeGuard: $bill \in AllBills$

tokenTypeGuard: $token \in TOKENS$

CR5

tokenAssignedToMeterGuard: $token = RegisteredTokens(meter)$

CR5

billAssignedToMeterGuard: $bill \in ran(\{meter\} \lhd MeterBill)$

CR10

then

setBillPaidAction: $BillPaid(bill) := TRUE$

end

Event AddMeterReading ⟨ordinary⟩ $\widehat{=}$

extends AddMeterReading

any

meter

reading

readingValue

token

where

meterTypeGuard: $meter \in RegisteredMeters$

readingTypeGuard: $reading \in READINGS$

currentWeekType: $CurrentWeek \in \mathbb{N}$

readingValueType: $readingValue \in \mathbb{N}$

readingIncreasesGuard: $\forall x, y \cdot x \in MeterReading[\{meter\}] \land$
$y \in ReadingValue[\{x\}] \Rightarrow readingValue \geq y$

CR4

tokenTypeGuard: $token \in TOKENS$

CR5

tokenAssignedToMeterGuard: $token = RegisteredTokens(meter)$

CR5

ReadingUniquenessCheck: $reading \notin AllReadings$

CR4/improvement in the spirit of ensuring Readings advance

then

AllReadingsAddAction: $AllReadings := AllReadings \cup \{reading\}$

AddMeterReadingAction: $MeterReading := MeterReading \cup \{meter \mapsto reading\}$

ReadingValueAction: $ReadingValue(reading) := readingValue$

ReadingReportedAction: $ReadingReported(reading) := CurrentWeek$

end

Event AdvanceTime ⟨ordinary⟩ $\widehat{=}$

extends AdvanceTime

when

billsGeneratedHoldGuard:

$(\forall meter, bill \cdot meter \in RegisteredMeters \land bill = MeterBill[\{meter\}]$
$\Rightarrow (BillGenerated \neq \varnothing \land (\exists bill2 \cdot bill2 \in dom(BillGenerated) \land bill2 \in bill \land$
$BillGenerated(bill2) = CurrentWeek)) \lor (CurrentWeek mod 4 \neq 0)) \lor (CurrentWeek =$
$0)$

CR8

noAdvanceHoldGuard:

$(\forall meter, x, y, z \cdot meter \in RegisteredMeters \land meter \in dom(MeterDisconnectState) \land$
$meter \in dom(MeterRegisteredDate) \land x = MeterBill[\{meter\}] \land y = (x \lhd$
$BillGenerated) \rhd \{item | item \in CurrentWeek - 999 .. CurrentWeek - 8\} \land z =$
$(dom(y) \lhd BillPaid) \Rightarrow (FALSE \in ran(z) \land ((MeterRegisteredDate(meter) +$
$12 \geq CurrentWeek) \lor MeterDisconnectState(meter) = TRUE)) \lor (FALSE \notin$
$ran(z))) \lor (\forall meter, x, y \cdot meter \in RegisteredMeters \land meter \in dom(MeterDisconnectState)$
$\land meter \in dom(MeterRegisteredDate) \land x = MeterReading[\{meter\}] \land y =$
$(x \lhd ReadingReported) \rhd \{item | item \in CurrentWeek - 4 .. CurrentWeek - 1\} \Rightarrow$
$(y = \varnothing \land (MeterRegisteredDate(meter) + 12 \geq CurrentWeek \lor MeterDisconnectState(meter) =$
$TRUE)) \lor (y \neq \varnothing))$

CR12, CR6

then

CurrentWeekAdvance: $CurrentWeek := CurrentWeek + 1$

end

Event DisconnectMeter ⟨ordinary⟩ $\widehat{=}$

extends DisconnectMeter

any

$meter$

where

meterTypeGuard: $meter \in RegisteredMeters \land meter \in dom(MeterRegisteredDate)$

disconnectCriteria: $(\forall x, y, z \cdot x = MeterBill[\{meter\}] \land y = (x \lhd BillGenerated) \rhd$
$\{item | item \in CurrentWeek - 999 .. CurrentWeek - 8\} \land z = (dom(y) \lhd$
$BillPaid) \Rightarrow (FALSE \in ran(z)) \land (MeterRegisteredDate(meter) + 12 \leq CurrentWeek)) \lor$
$(\forall x, y \cdot x = MeterReading[\{meter\}] \land y = (x \lhd ReadingReported) \rhd \{item | item \in$
$CurrentWeek - 4 .. CurrentWeek - 1\} \Rightarrow y = \varnothing \land MeterRegisteredDate(meter) +$
$12 \leq CurrentWeek)$

meterNotAlreadyDisconnected: $meter \in dom(MeterDisconnectState)$
$\land MeterDisconnectState(meter) \neq TRUE$

then

DisconnectMeterAction: $MeterDisconnectState(meter) := TRUE$

end

END

Appendix C

Drone case study - initial formal model - machine & context

CONTEXT DroneContextInitial

SETS

LOCATIONS

DRONES

POINTS

VALIDATIONSTATE

CONSTANTS

VALID

INVALID

UNDETERMINED

HOME

AXIOMS

ValidationStatePartitionAxm: $partition(VALIDATIONSTATE, \{VALID\}, \{INVALID\}, \{UNDETERMINED\})$

HomeLocationAxm: $HOME \in LOCATIONS$

END

MACHINE DroneMachineInitial

SEES DroneContextInitial

VARIABLES

RegisteredAircraft

RestrictedLocations

Plans

Commands

AircraftLocations

AircraftRoutes

PlanValidationState

CommandValidationState

CancelledCommands

INVARIANTS

TypeRegisteredAircraft: $RegisteredAircraft \subseteq DRONES$

TypeRestrictedLocations: $RestrictedLocations \subseteq LOCATIONS$

TypePlans: $Plans \subseteq \mathbb{P}(LOCATIONS)$

TypeCommands: $Commands \subseteq \mathbb{P}(LOCATIONS)$

TypeAircraftLocations: $AircraftLocations \in RegisteredAircraft \pfun LOCATIONS$

TypeAircraftRoutes: $AircraftRoutes \in RegisteredAircraft \pfun (Commands \cup \{\{HOME\}\} \cup \{\varnothing\})$

TypePlanValidationState: $PlanValidationState \in Plans \pfun VALIDATIONSTATE$

TypeCommandValidationState: $CommandValidationState \in Commands \pfun VALIDATIONSTATE$

AircraftRoutesDomainInv: $dom(AircraftLocations) = RegisteredAircraft$

AircraftRoutesRangeInv: $ran(AircraftLocations) \subseteq LOCATIONS$

CancelledCommandsType: $CancelledCommands \subseteq \mathbb{P}(LOCATIONS)$

EVENTS

Initialisation

begin

InitRegisteredAircraft: $RegisteredAircraft := \varnothing$

InitRestrictedLocations: $RestrictedLocations := \varnothing$

InitPlans: $Plans := \{\{HOME\}, \varnothing\}$

Home is always a valid plan.

InitCommands: $Commands := \{\{HOME\}, \varnothing\}$

Home is always a valid command.

InitAircraftLocations: $AircraftLocations := \varnothing$

InitAircraftRoutes: $AircraftRoutes := \varnothing$

InitCommandValidationState: $CommandValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitPlanValidationState: $PlanValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitCancelledCommands: $CancelledCommands := \varnothing$

end

Event RegisterAircraft ⟨ordinary⟩ $\widehat{=}$

any

aircraft

where

aircraftType: $aircraft \in DRONES$

aircraftNotDuplicate: $aircraft \notin RegisteredAircraft$

then

addAircraft: $RegisteredAircraft := RegisteredAircraft \cup \{aircraft\}$

addAircraftLocations: $AircraftLocations := AircraftLocations \cup \{aircraft \mapsto HOME\}$

addAircraftRoute: $AircraftRoutes := AircraftRoutes \cup \{aircraft \mapsto \varnothing\}$

end

Event RegisterRestrictedLocation ⟨ordinary⟩ $\widehat{=}$

any

restricted

where

restrictedType: $restricted \in LOCATIONS$

restrictedNotDuplicate: $restricted \notin RestrictedLocations$

restrictedNotHome: $restricted \neq HOME$

then

addRestricted: $RestrictedLocations := RestrictedLocations \cup \{restricted\}$

invalidExistingCommands:

$CommandValidationState := (\{y \cdot y \in dom(CommandValidationState) \land restricted \in y | y\} \lhd CommandValidationState) \cup$
$(\lambda q \cdot q \in dom(CommandValidationState) \land restricted \in q \land CommandValidationState(q) \neq INVALID | INVALID)$

replaceAffectedAircraft:

$AircraftRoutes := (AircraftRoutes \rhd \{y \cdot y \in ran(AircraftRoutes) \land restricted \in y | y\}) \cup$
$(\lambda q \cdot q \in dom(AircraftRoutes) \land q \in dom(AircraftLocations) \land AircraftLocations(q) = HOME \land restricted \in AircraftRoutes(q) | \varnothing) \cup$
$(\lambda q \cdot q \in dom(AircraftRoutes) \land q \in dom(AircraftLocations) \land AircraftLocations(q) \neq HOME \land restricted \in AircraftRoutes(q) | \{HOME\})$

end

Event TransmitPlan ⟨ordinary⟩ $\widehat{=}$

any

plan

where

locationsType: $plan \in \mathbb{P}(LOCATIONS)$

planNotDuplicate: $plan \notin Plans$

then

addPlan: $Plans := Plans \cup \{plan\}$

planValidationState: $PlanValidationState(plan) := UNDETERMINED$

Added to discharge TypePlanValidationState invariant, or else it would be unprovable.

end

Event ValidatePlan ⟨ordinary⟩ $\widehat{=}$

any

plan

where

planType: $plan \subseteq LOCATIONS$

planRegistered: $plan \in Plans$

planDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin plan$

then

validateAction: $PlanValidationState(plan) := VALID$

end

Event TransmitCommand ⟨ordinary⟩ $\widehat{=}$

any

command

where

commandType: $command \subseteq LOCATIONS$

commandNotDuplicate: $command \notin Commands$

commandContainsHome: $HOME \in command$

then

addCommand: $Commands := Commands \cup \{command\}$

commandValidationState: $CommandValidationState(command) := UNDETERMINED$

end

Event ValidateCommand ⟨ordinary⟩ $\widehat{=}$

any

command

where

commandType: $command \subseteq LOCATIONS$

commandRegistered: $command \in Commands$

commandDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin command$

then

commandValidationStateUpdate: $CommandValidationState(command) := VALID$

end

Event TaskAircraft ⟨ordinary⟩ $\widehat{=}$

any

aircraft

command

where

aircraftType: $aircraft \in RegisteredAircraft$

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

then

taskAircraftGo: $AircraftRoutes(aircraft) := command$

end

Event RetaskAircraft ⟨ordinary⟩ $\widehat{=}$

any

command

aircraft

where

commandType: $command \in Commands$

aircraftType: $aircraft \in RegisteredAircraft$

then

retaskAircraftGo: $AircraftRoutes(aircraft) := command$

end

Event RecallAircraft ⟨ordinary⟩ $\widehat{=}$

any

aircraft

where

aircraftType: $aircraft \in RegisteredAircraft$

aircraftTypeHard: $aircraft \in DRONES$

aircraftLocationCheck: $AircraftLocations(aircraft) \neq HOME$

then

aircraftRouteChange: $AircraftRoutes(aircraft) := \{HOME\}$

end

Event CancelCommand ⟨ordinary⟩ $\widehat{=}$

any

command

where

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

then

removeCommands: $Commands := Commands \setminus \{command\}$

removeCommandValidationState: $CommandValidationState := \{command\} \lhd CommandValidationState$

removeAircraftRoutes: $AircraftRoutes := (\lambda q \cdot q \in dom(AircraftRoutes) \land AircraftRoutes(q) = command | \{HOME\})$

addToCancelled: $CancelledCommands := CancelledCommands \cup \{command\}$

end

Event ReportLocation ⟨ordinary⟩ $\widehat{=}$

any

aircraft

newlocation

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

end

Event ReportDeviation ⟨ordinary⟩ $\widehat{=}$

any

deviation

aircraft

where

deviationType: $deviation \in LOCATIONS$

aircraftType: $aircraft \in RegisteredAircraft$

then

updateLocationDeviation: $AircraftLocations(aircraft) := deviation$

end

Event ConfirmReceipt ⟨ordinary⟩ $\widehat{=}$

any

route_or_update

where

RouteOrUpdateType: $route\_or\_update \in \mathbb{P}(LOCATIONS)$

RouteOrUpdateBelongs: $route\_or\_update \in Commands$

then

skip

end

Event ConfirmCancellation ⟨ordinary⟩ $\widehat{=}$

any

command

result

where

commandType: $command \in \mathbb{P}(LOCATIONS)$

commandInCancelled: $command \in CancelledCommands$

resultsSet: $result = TRUE$

then

skip

end

Event QueryPlanValidationState ⟨ordinary⟩ $\widehat{=}$

any

plan

result

where

planBelongsToPlans: $plan \in (Plans \setminus \{\varnothing, \{HOME\}\})$

planDomain: $plan \in dom(PlanValidationState)$

resultsSet: $result = PlanValidationState(plan)$

then

skip

end

Event QueryCommandValidationState ⟨ordinary⟩ $\widehat{=}$

any

command

result

where

commandBelongsToCommands: $command \in (Commands \setminus \{\varnothing, \{HOME\}\})$

commandDomain: $command \in dom(CommandValidationState)$

resultsSet: $result = CommandValidationState(command)$

then

skip

end

END

Appendix D

Drone case study - first refinement of formal model - machine & context

CONTEXT DroneContext2
EXTENDS DroneContextInitial
END

MACHINE DroneMachine2

REFINES DroneMachineInitial

SEES DroneContext2

VARIABLES

RegisteredAircraft

RestrictedLocations

Plans

Commands

AircraftLocations

AircraftRoutes

PlanValidationState

CommandValidationState

CancelledCommands

PlanValidationQueue [CR1

CommandValidationQueue [CR3

PlanValidationQueueCount [CR1

CommandValidationQueueCount [CR3

RecentTimestamp

INVARIANTS

`PlanValidationQueueType`: $PlanValidationQueue \in \mathbb{P}(LOCATIONS) \rightarrowtail 0 \mathinner{..} PlanValidationQueueCount - 1$

`CommandValidationQueueType`: $CommandValidationQueue \in \mathbb{P}(LOCATIONS) \rightarrowtail 0 \mathinner{..} CommandValidationQueueCount - 1$

`PlanValidationQueueCountType`: $PlanValidationQueueCount \in \mathbb{N}$

`CommandValidationQueueCountType`: $CommandValidationQueueCount \in \mathbb{N}$

`RecentTimestampType`: $RecentTimestamp \in RegisteredAircraft \rightarrow \mathbb{N}$

EVENTS

Initialisation ⟨extended⟩

begin

`InitRegisteredAircraft`: $RegisteredAircraft := \varnothing$

`InitRestrictedLocations`: $RestrictedLocations := \varnothing$

`InitPlans`: $Plans := \{\{HOME\}, \varnothing\}$

Home is always a valid plan.

`InitCommands`: $Commands := \{\{HOME\}, \varnothing\}$

Home is always a valid command.

`InitAircraftLocations`: $AircraftLocations := \varnothing$

`InitAircraftRoutes`: $AircraftRoutes := \varnothing$

`InitCommandValidationState`: $CommandValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitPlanValidationState: $PlanValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitCancelledCommands: $CancelledCommands := \varnothing$

PlanValidationQueueInit: $PlanValidationQueue := \varnothing$

CommandValidationQueueInit: $CommandValidationQueue := \varnothing$

PlanValidationQueueCountInit: $PlanValidationQueueCount := 0$

CommandValidationQueueCountInit: $CommandValidationQueueCount := 0$

RecentTimestampInit: $RecentTimestamp := \varnothing$

end

Event RegisterAircraft ⟨ordinary⟩ $\widehat{=}$

extends RegisterAircraft

any

$aircraft$

where

aircraftType: $aircraft \in DRONES$

aircraftNotDuplicate: $aircraft \notin RegisteredAircraft$

then

addAircraft: $RegisteredAircraft := RegisteredAircraft \cup \{aircraft\}$

addAircraftLocations: $AircraftLocations := AircraftLocations \cup \{aircraft \mapsto HOME\}$

addAircraftRoute: $AircraftRoutes := AircraftRoutes \cup \{aircraft \mapsto \varnothing\}$

act1: $RecentTimestamp := RecentTimestamp \cup \{aircraft \mapsto 0\}$

end

Event RegisterRestrictedLocation ⟨ordinary⟩ $\widehat{=}$

extends RegisterRestrictedLocation

any

$restricted$

where

restrictedType: $restricted \in LOCATIONS$

restrictedNotDuplicate: $restricted \notin RestrictedLocations$

restrictedNotHome: $restricted \neq HOME$

then

addRestricted: $RestrictedLocations := RestrictedLocations \cup \{restricted\}$

invalidExistingCommands:

$CommandValidationState := (\{y \cdot y \in dom(CommandValidationState) \land restricted \in y | y\} \lhd CommandValidationState) \cup$
$(\lambda q \cdot q \in dom(CommandValidationState) \land restricted \in q$
$\land CommandValidationState(q) \neq INVALID | INVALID)$

replaceAffectedAircraft:

$AircraftRoutes := (AircraftRoutes \rhd \{y \cdot y \in ran(AircraftRoutes) \land restricted \in y | y\}) \cup$

$(\lambda q \cdot q \in dom(AircraftRoutes) \land q \in dom(AircraftLocations) \land AircraftLocations(q) = HOME \land restricted \in AircraftRoutes(q) | \varnothing) \cup$
$(\lambda q \cdot q \in dom(AircraftRoutes) \land q \in dom(AircraftLocations) \land AircraftLocations(q) \neq HOME \land restricted \in AircraftRoutes(q) | \{HOME\})$

end

Event TransmitPlan ⟨ordinary⟩ $\widehat{=}$

extends TransmitPlan

any

$plan$

where

locationsType: $plan \in \mathbb{P}(LOCATIONS)$

planNotDuplicate: $plan \notin Plans$

planValidationQueueGuard: $PlanValidationQueueCount \in \mathbb{N}$

grd1: $plan \cap RestrictedLocations = \varnothing$

then

addPlan: $Plans := Plans \cup \{plan\}$

planValidationState: $PlanValidationState(plan) := UNDETERMINED$

PlanAddQueue: $PlanValidationQueue(plan) := PlanValidationQueueCount$

PlanValidationQueueIncrement: $PlanValidationQueueCount := PlanValidationQueueCount + 1$

end

Event ValidatePlan ⟨ordinary⟩ $\widehat{=}$

extends ValidatePlan

any

$plan$

where

planType: $plan \subseteq LOCATIONS$

planRegistered: $plan \in Plans$

planDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin plan$

grd1: $dom(\{plan\} \lhd PlanValidationQueue) \subseteq \mathbb{P}(LOCATIONS)$

grd2: $0 \in ran(PlanValidationQueue)$

grd3: $PlanValidationQueue^{-1}(0) = plan$

then

validateAction: $PlanValidationState(plan) := VALID$

act1: $PlanValidationQueue := (\lambda q \cdot q \in dom(\{plan\} \lhd PlanValidationQueue) \land q \in \mathbb{P}(LOCATIONS) \land PlanValidationQueue(q) \in 0 \,..\, PlanValidationQueueCount | PlanValidationQueue(q) - 1)$

act2: $PlanValidationQueueCount := PlanValidationQueueCount - 1$

end

Event TransmitCommand ⟨ordinary⟩ $\widehat{=}$

extends TransmitCommand

any

$command$

where

commandType: $command \subseteq LOCATIONS$

commandNotDuplicate: $command \notin Commands$

commandContainsHome: $HOME \in command$

grd1: $CommandValidationQueueCount \in \mathbb{N}$

grd2: $command \cap RestrictedLocations = \varnothing$

then

addCommand: $Commands := Commands \cup \{command\}$

commandValidationState: $CommandValidationState(command) := UNDETERMINED$

act1: $CommandValidationQueue(command) := CommandValidationQueueCount$

act2: $CommandValidationQueueCount := CommandValidationQueueCount + 1$

end

Event ValidateCommand ⟨ordinary⟩ $\widehat{=}$

extends ValidateCommand

any

$command$

where

commandType: $command \subseteq LOCATIONS$

commandRegistered: $command \in Commands$

commandDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin command$

grd1: $dom(\{command\} \lhd CommandValidationQueue) \subseteq \mathbb{P}(LOCATIONS)$

grd2: $0 \in ran(CommandValidationQueue)$

grd3: $CommandValidationQueue^{-1}(0) = command$

then

commandValidationStateUpdate: $CommandValidationState(command) := VALID$

act1: $CommandValidationQueue := (\lambda q \cdot q \in dom(\{command\} \lhd CommandValidationQueue) \land q \in \mathbb{P}(LOCATIONS) | CommandValidationQueue(q) - 1)$

act2: $CommandValidationQueueCount := CommandValidationQueueCount - 1$

end

Event TaskAircraft ⟨ordinary⟩ $\widehat{=}$

extends TaskAircraft

any

aircraft

command

where

aircraftType: $aircraft \in RegisteredAircraft$

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

then

taskAircraftGo: $AircraftRoutes(aircraft) := command$

end

Event RetaskAircraft ⟨ordinary⟩ $\widehat{=}$

extends RetaskAircraft

any

command

aircraft

where

commandType: $command \in Commands$

aircraftType: $aircraft \in RegisteredAircraft$

then

retaskAircraftGo: $AircraftRoutes(aircraft) := command$

end

Event RecallAircraft ⟨ordinary⟩ $\widehat{=}$

extends RecallAircraft

any

aircraft

where

aircraftType: $aircraft \in RegisteredAircraft$

aircraftTypeHard: $aircraft \in DRONES$

aircraftLocationCheck: $AircraftLocations(aircraft) \neq HOME$

then

aircraftRouteChange: $AircraftRoutes(aircraft) := \{HOME\}$

end

Event CancelCommand ⟨ordinary⟩ $\widehat{=}$

extends CancelCommand

any

command

where

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

then

removeCommands: $Commands := Commands \setminus \{command\}$

removeCommandValidationState: $CommandValidationState := \{command\} \lhd CommandValidationState$

removeAircraftRoutes: $AircraftRoutes := (\lambda q \cdot q \in dom(AircraftRoutes) \land AircraftRoutes(q) = command|\{HOME\})$

addToCancelled: $CancelledCommands := CancelledCommands \cup \{command\}$

end

Event ReportLocation ⟨ordinary⟩ $\widehat{=}$

extends ReportLocation

any

$aircraft$

$newlocation$

timestamp

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

grd1: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

end

Event ReportDeviation ⟨ordinary⟩ $\widehat{=}$

extends ReportDeviation

any

$deviation$

$aircraft$

timestamp

where

deviationType: $deviation \in LOCATIONS$

aircraftType: $aircraft \in RegisteredAircraft$

timestampGuard: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

then

updateLocationDeviation: $AircraftLocations(aircraft) := deviation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

end

Event ConfirmReceipt ⟨ordinary⟩ $\widehat{=}$

extends ConfirmReceipt

any

$route\_or\_update$

where

RouteOrUpdateType: $route\_or\_update \in \mathbb{P}(LOCATIONS)$

RouteOrUpdateBelongs: $route\_or\_update \in Commands$

then

skip

end

Event ConfirmCancellation ⟨ordinary⟩ $\widehat{=}$

extends ConfirmCancellation

any

command

result

where

commandType: $command \in \mathbb{P}(LOCATIONS)$

commandInCancelled: $command \in CancelledCommands$

resultsSet: $result = TRUE$

then

skip

end

Event QueryPlanValidationState ⟨ordinary⟩ $\widehat{=}$

extends QueryPlanValidationState

any

plan

result

where

planBelongsToPlans: $plan \in (Plans \setminus \{\varnothing, \{HOME\}\})$

planDomain: $plan \in dom(PlanValidationState)$

resultsSet: $result = PlanValidationState(plan)$

then

skip

end

Event QueryCommandValidationState ⟨ordinary⟩ $\widehat{=}$

extends QueryCommandValidationState

any

command

result

where

commandBelongsToCommands: $command \in (Commands \setminus \{\varnothing, \{HOME\}\})$

commandDomain: $command \in dom(CommandValidationState)$

resultsSet: $result = CommandValidationState(command)$

then

skip

end

END

Appendix E

Drone case study - second refinement of formal model - machine & context

CONTEXT DroneContext3

EXTENDS DroneContext2

SETS

COMMANDSTATE

ROUTEIDENTIFIERS

CONSTANTS

UNASSIGNED

ASSIGNED

COMPLETE

SPECIAL

CONFIRMED

NO_ROUTE

HOME_ONLY_ROUTE

AXIOMS

commandstatepartitionaxm: $partition(COMMANDSTATE, \{UNASSIGNED\}, \{ASSIGNED\}, \{CONFIRMED\}, \{COMPLETE\}, \{SPECIAL\})$

norouteaxm: $\{NO\_ROUTE, HOME\_ONLY\_ROUTE\} \subseteq ROUTEIDENTIFIERS$

axm1: $NO\_ROUTE \neq HOME\_ONLY\_ROUTE$

END

MACHINE DroneMachine3

REFINES DroneMachine2

SEES DroneContext3

VARIABLES

RegisteredAircraft

RestrictedLocations

Plans

Commands

AircraftLocations

AircraftRoutes

PlanValidationState

CommandValidationState

CancelledCommands

PlanValidationQueue [CR1

CommandValidationQueue [CR3

PlanValidationQueueCount [CR1

CommandValidationQueueCount [CR3

RecentTimestamp

CommandStatus [CR4, CR5

AircraftLocationHistory

RouteIDs

AircraftRouteIDs

AircraftRouteTasks

CurrentIdentifiers

INVARIANTS

`CommandStatusType`: $CommandStatus \in Commands \rightarrow COMMANDSTATE$

`AircraftLocationHistoryType`: $AircraftLocationHistory \in RegisteredAircraft \rightarrow \mathbb{P}(LOCATIONS)$

`CurrentIdentifiersType`: $CurrentIdentifiers \subseteq ROUTEIDENTIFIERS$

`RouteIdentifierType`: $RouteIDs \in Commands \rightarrow CurrentIdentifiers$

`AircraftRouteIDsType`: $AircraftRouteIDs \in RegisteredAircraft \rightarrow CurrentIdentifiers$

`AircraftRouteTasksType`: $AircraftRouteTasks \in CurrentIdentifiers \nrightarrow \mathbb{P}(LOCATIONS)$

EVENTS

Initialisation ⟨extended⟩

begin

`InitRegisteredAircraft`: $RegisteredAircraft := \varnothing$

InitRestrictedLocations: $RestrictedLocations := \varnothing$

InitPlans: $Plans := \{\{HOME\}, \varnothing\}$

Home is always a valid plan.

InitCommands: $Commands := \{\{HOME\}, \varnothing\}$

Home is always a valid command.

InitAircraftLocations: $AircraftLocations := \varnothing$

InitAircraftRoutes: $AircraftRoutes := \varnothing$

InitCommandValidationState: $CommandValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitPlanValidationState: $PlanValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitCancelledCommands: $CancelledCommands := \varnothing$

PlanValidationQueueInit: $PlanValidationQueue := \varnothing$

CommandValidationQueueInit: $CommandValidationQueue := \varnothing$

PlanValidationQueueCountInit: $PlanValidationQueueCount := 0$

CommandValidationQueueCountInit: $CommandValidationQueueCount := 0$

RecentTimestampInit: $RecentTimestamp := \varnothing$

CommandStatusInit: $CommandStatus := \{\{HOME\} \mapsto SPECIAL, \varnothing \mapsto SPECIAL\}$

AircraftLocationHistoryInit: $AircraftLocationHistory := \varnothing$

CurrentIdentifiersInit: $CurrentIdentifiers := \{HOME\_ONLY\_ROUTE, NO\_ROUTE\}$

RouteIdentifierInit: $RouteIDs := \{\{HOME\} \mapsto HOME\_ONLY\_ROUTE, \varnothing \mapsto NO\_ROUTE\}$

AircraftRouteIDsInit: $AircraftRouteIDs := \varnothing$

AircraftRouteTasksInit: $AircraftRouteTasks := \{HOME\_ONLY\_ROUTE \mapsto \{HOME\}, NO\_ROUTE \mapsto \varnothing\}$

end

Event RegisterAircraft ⟨ordinary⟩ $\widehat{=}$

extends RegisterAircraft

any

$aircraft$

where

aircraftType: $aircraft \in DRONES$

aircraftNotDuplicate: $aircraft \notin RegisteredAircraft$

norouteCurrent: $NO\_ROUTE \in CurrentIdentifiers$

then

addAircraft: $RegisteredAircraft := RegisteredAircraft \cup \{aircraft\}$

addAircraftLocations: $AircraftLocations := AircraftLocations \cup \{aircraft \mapsto HOME\}$

addAircraftRoute: $AircraftRoutes := AircraftRoutes \cup \{aircraft \mapsto \varnothing\}$

act1: $RecentTimestamp := RecentTimestamp \cup \{aircraft \mapsto 0\}$

AircraftLocationHistoryRegister: $AircraftLocationHistory(aircraft) := \{HOME\}$

AircraftRouteIDsRegister: $AircraftRouteIDs := AircraftRouteIDs \cup \{aircraft \mapsto NO\_ROUTE\}$

end

Event RegisterRestrictedLocation ⟨ordinary⟩ $\widehat{=}$

extends RegisterRestrictedLocation

any

restricted

aircraft_not_at_home

aircraft_at_home

aircraft_with_restricted_waypoints

where

restrictedType: $restricted \in LOCATIONS$

restrictedNotDuplicate: $restricted \notin RestrictedLocations$

restrictedNotHome: $restricted \neq HOME$

grd1: $\{HOME\_ONLY\_ROUTE, NO\_ROUTE\} \subseteq CurrentIdentifiers$

grd2: $aircraft\_not\_at\_home = \{y \cdot y \in RegisteredAircraft \wedge y \in dom(AircraftLocations) \wedge AircraftLocations(y) \neq HOME | y\}$

grd3: $aircraft\_at\_home = \{y \cdot y \in RegisteredAircraft \wedge y \in dom(AircraftLocations) \wedge AircraftLocations(y) = HOME | y\}$

grd4: $aircraft\_with\_restricted\_waypoints = \{y \cdot y \in (aircraft\_at\_home \cup aircraft\_not\_at\_home) \wedge y \in dom(AircraftRoutes) \wedge restricted \in AircraftRoutes(y) | y\}$

then

addRestricted: $RestrictedLocations := RestrictedLocations \cup \{restricted\}$

invalidExistingCommands:

$CommandValidationState := (\{y \cdot y \in dom(CommandValidationState) \wedge restricted \in y | y\} \lhd CommandValidationState) \cup (\lambda q \cdot q \in dom(CommandValidationState) \wedge restricted \in q \wedge CommandValidationState(q) \neq INVALID | INVALID)$

replaceAffectedAircraft:

$AircraftRoutes := (AircraftRoutes \rhd \{y \cdot y \in ran(AircraftRoutes) \wedge restricted \in y | y\}) \cup (\lambda q \cdot q \in dom(AircraftRoutes) \wedge q \in dom(AircraftLocations) \wedge AircraftLocations(q) = HOME \wedge restricted \in AircraftRoutes(q) | \varnothing) \cup (\lambda q \cdot q \in dom(AircraftRoutes) \wedge q \in dom(AircraftLocations) \wedge AircraftLocations(q) \neq HOME \wedge restricted \in AircraftRoutes(q) | \{HOME\})$

act1:

$AircraftRouteIDs := (aircraft\_with\_restricted\_waypoints \lhd AircraftRouteIDs) \cup \{y \cdot y \in aircraft\_with\_restricted\_waypoints \cap aircraft\_at\_home | y \mapsto NO\_ROUTE\}$

$\cup \{y \cdot y \in aircraft\_with\_restricted\_waypoints \cap aircraft\_not\_at\_home | y \mapsto HOME\_ONLY\_ROUTE\}$

end

Event TransmitPlan ⟨ordinary⟩ $\widehat{=}$

extends TransmitPlan

any

$plan$

where

locationsType: $plan \in \mathbb{P}(LOCATIONS)$

planNotDuplicate: $plan \notin Plans$

planValidationQueueGuard: $PlanValidationQueueCount \in \mathbb{N}$

grd1: $plan \cap RestrictedLocations = \varnothing$

then

addPlan: $Plans := Plans \cup \{plan\}$

planValidationState: $PlanValidationState(plan) := UNDETERMINED$

PlanAddQueue: $PlanValidationQueue(plan) := PlanValidationQueueCount$

PlanValidationQueueIncrement: $PlanValidationQueueCount := PlanValidationQueueCount + 1$

end

Event ValidatePlan ⟨ordinary⟩ $\widehat{=}$

extends ValidatePlan

any

$plan$

where

planType: $plan \subseteq LOCATIONS$

planRegistered: $plan \in Plans$

planDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin plan$

grd1: $dom(\{plan\} \lhd PlanValidationQueue) \subseteq \mathbb{P}(LOCATIONS)$

grd2: $0 \in ran(PlanValidationQueue)$

grd3: $PlanValidationQueue^{-1}(0) = plan$

then

validateAction: $PlanValidationState(plan) := VALID$

act1: $PlanValidationQueue := (\lambda q \cdot q \in dom(\{plan\} \lhd PlanValidationQueue) \land q \in \mathbb{P}(LOCATIONS) \land PlanValidationQueue(q) \in 0..PlanValidationQueueCount | PlanValidationQueue(q) - 1)$

act2: $PlanValidationQueueCount := PlanValidationQueueCount - 1$

end

Event TransmitCommand ⟨ordinary⟩ $\widehat{=}$

extends TransmitCommand

any

$command$

identifier

where

commandType: $command \subseteq LOCATIONS$

commandNotDuplicate: $command \notin Commands$

commandContainsHome: $HOME \in command$

grd1: $CommandValidationQueueCount \in \mathbb{N}$

grd2: $command \cap RestrictedLocations = \varnothing$

identifierType: $identifier \in ROUTEIDENTIFIERS$

identifierIsNew: $identifier \notin CurrentIdentifiers$

commandNotIn: $command \neq \{HOME\} \land command \neq \varnothing$

then

addCommand: $Commands := Commands \cup \{command\}$

commandValidationState: $CommandValidationState(command) := UNDETERMINED$

act1: $CommandValidationQueue(command) := CommandValidationQueueCount$

act2: $CommandValidationQueueCount := CommandValidationQueueCount + 1$

act3: $CommandStatus(command) := UNASSIGNED$

act4: $RouteIDs := RouteIDs \cup \{command \mapsto identifier\}$

act5: $CurrentIdentifiers := CurrentIdentifiers \cup \{identifier\}$

end

Event ValidateCommand ⟨ordinary⟩ $\widehat{=}$

extends ValidateCommand

any

$command$

where

commandType: $command \subseteq LOCATIONS$

commandRegistered: $command \in Commands$

commandDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin command$

grd1: $dom(\{command\} \lhd CommandValidationQueue) \subseteq \mathbb{P}(LOCATIONS)$

grd2: $0 \in ran(CommandValidationQueue)$

grd3: $CommandValidationQueue^{-1}(0) = command$

then

commandValidationStateUpdate: $CommandValidationState(command) := VALID$

act1: $CommandValidationQueue := (\lambda q \cdot q \in dom(\{command\} \lhd CommandValidationQueue) \land q \in \mathbb{P}(LOCATIONS) | CommandValidationQueue(q) - 1)$

act2: $CommandValidationQueueCount := CommandValidationQueueCount - 1$

end

Event TaskAircraft ⟨ordinary⟩ $\widehat{=}$

extends TaskAircraft

any

$aircraft$

$command$

where

aircraftType: $aircraft \in RegisteredAircraft$

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

commandDomainCheck: $command \in dom(CommandValidationState)$

commandCheck: $CommandValidationState(command) = VALID$

aircraftRouteDom: $aircraft \in dom(AircraftRoutes)$

aircraftNoRoute:

$AircraftRoutes(aircraft) = \varnothing \lor$
$(AircraftRoutes(aircraft) \neq \varnothing \land$
$AircraftRoutes(aircraft) \in Commands \land$
$CommandStatus(AircraftRoutes(aircraft)) \in \{COMPLETE, SPECIAL\})$

routeNotSpecial: $CommandStatus(command) = UNASSIGNED$

aircraftRouteIDCheck: $RouteIDs(command) \notin \{NO\_ROUTE, HOME\_ONLY\_ROUTE\}$

noExistingAssignment: $RouteIDs(command) \notin ran(AircraftRouteIDs)$

then

taskAircraftGo: $AircraftRoutes(aircraft) := command$

CommandStatusUpdate: $CommandStatus(command) := ASSIGNED$

act1: $AircraftRouteIDs(aircraft) := RouteIDs(command)$

act2: $AircraftRouteTasks(RouteIDs(command)) := command$

end

Event RetaskAircraft ⟨ordinary⟩ $\widehat{=}$

extends RetaskAircraft

any

$command$

$aircraft$

existing_command

where

commandType: $command \in Commands$

aircraftType: $aircraft \in RegisteredAircraft$

commandDomainCheck: $command \in dom(CommandValidationState)$

commandCheck: $CommandValidationState(command) = VALID$

aircraftRouteDom: $aircraft \in dom(AircraftRoutes)$

existingCommandCheck: $existing\_command = AircraftRoutes(aircraft)$

existingNotBlank: $existing\_command \neq \varnothing$

existingNotSame: $existing\_command \neq command$

existingNotSameTwo: $command \neq \{HOME\} \land command \neq \varnothing$

existingIsCommand:

$existing\_command \in Commands \land existing\_command \in dom(CommandStatus)$

$\land CommandStatus(existing\_command) \notin \{COMPLETE, SPECIAL\}$

CommandStateCheck: $CommandStatus(command) = UNASSIGNED$

then

retaskAircraftGo: $AircraftRoutes(aircraft) := command$

CommandStatusExistingUpdate: $CommandStatus := (\{command, existing\_command\} \lhd CommandStatus) \cup \{command \mapsto ASSIGNED\} \cup \{existing\_command \mapsto UNASSIGNED\}$

act2: $AircraftRouteIDs(aircraft) := RouteIDs(command)$

act3: $AircraftRouteTasks(RouteIDs(command)) := command$

end

Event RecallAircraft ⟨ordinary⟩ $\widehat{=}$

extends RecallAircraft

any

$aircraft$

where

aircraftType: $aircraft \in RegisteredAircraft$

aircraftTypeHard: $aircraft \in DRONES$

aircraftLocationCheck: $AircraftLocations(aircraft) \neq HOME$

grd1: $\{HOME\} \in dom(RouteIDs)$

then

aircraftRouteChange: $AircraftRoutes(aircraft) := \{HOME\}$

act1: $AircraftRouteIDs(aircraft) := RouteIDs(\{HOME\})$

act2: $AircraftRouteTasks(RouteIDs(\{HOME\})) := \{HOME\}$

end

Event CancelCommand ⟨ordinary⟩ $\widehat{=}$

extends CancelCommand

any

$command$

where

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

grd1: $CommandStatus(command) \neq COMPLETE$

grd3: $HOME\_ONLY\_ROUTE \in CurrentIdentifiers$

grd4: $command \in dom(RouteIDs)$

then

removeCommands: $Commands := Commands \setminus \{command\}$

removeCommandValidationState: $CommandValidationState := \{command\} \lhd CommandValidationState$

removeAircraftRoutes: $AircraftRoutes := (\lambda q \cdot q \in dom(AircraftRoutes) \land AircraftRoutes(q) = command | \{HOME\})$

addToCancelled: $CancelledCommands := CancelledCommands \cup \{command\}$

CommandStatusRemove: $CommandStatus := \{command\} \lhd CommandStatus$

AircraftRouteIDsChange: $AircraftRouteIDs := (AircraftRouteIDs \rhd \{RouteIDs(command)\}) \cup (\lambda q \cdot q \in RegisteredAircraft \land q \in dom(AircraftRouteIDs) \land AircraftRouteIDs(q) = RouteIDs(command) | HOME\_ONLY\_ROUTE)$

RouteIDsRemove: $RouteIDs := \{command\} \lhd RouteIDs$

end

Event ReportLocationElsewhere ⟨ordinary⟩ $\widehat{=}$

extends ReportLocation

any

$aircraft$

$newlocation$

$timestamp$

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

grd1: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

grd3: $aircraft \in dom(AircraftLocationHistory)$

grd5: $aircraft \in dom(AircraftRoutes)$

grd4: $newlocation \in AircraftRoutes(aircraft)$

grd6: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land CommandStatus(AircraftRoutes(aircraft)) = CONFIRMED$

grd7: $AircraftRoutes(aircraft) \neq \{HOME\}$

grd8: $aircraft \in dom(AircraftRouteIDs) \land AircraftRouteIDs(aircraft) \in dom(AircraftRouteTasks)$

grd9: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \in \mathbb{P}(LOCATIONS)$

grd10: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{HOME\} \neq \varnothing$

grd11: $newlocation \neq HOME$

grd12: $newlocation \in AircraftRouteTasks(AircraftRouteIDs(aircraft))$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

AircraftLocationHistoryUpdate: $AircraftLocationHistory(aircraft) := (AircraftLocationHistory(aircraft) \cup \{newlocation\})$

AircraftRouteTasksUpdate: $AircraftRouteTasks := AircraftRouteTasks \Leftarrow \{AircraftRouteIDs(aircraft) \mapsto AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{newlocation\}\}$

end

Event ReportLocationHomeNormal ⟨ordinary⟩ $\widehat{=}$

extends ReportLocation

any

$aircraft$

$newlocation$

$timestamp$

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

grd1: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

grd2: $aircraft \in dom(AircraftLocationHistory)$

grd3: $aircraft \in dom(AircraftRoutes)$

grd5: $newlocation \in AircraftRoutes(aircraft)$

grd4: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land CommandStatus(AircraftRoutes(aircraft)) \in \{CONFIRMED\}$

grd7: $aircraft \in dom(AircraftRouteIDs) \land AircraftRouteIDs(aircraft) \in dom(AircraftRouteTasks)$

grd6: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \in \mathbb{P}(LOCATIONS)$

grd8: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{HOME\} = \varnothing$

grd9: $newlocation \in AircraftRouteTasks(AircraftRouteIDs(aircraft))$

grd10: $NO\_ROUTE \in CurrentIdentifiers$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

AircraftLocationHistoryUpdate: $AircraftLocationHistory(aircraft) := (AircraftLocationHistory(aircraft) \cup \{newlocation\})$

AircraftRouteTasksUpdate: $AircraftRouteTasks := AircraftRouteTasks \Leftarrow \{AircraftRouteIDs(aircraft) \mapsto AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{newlocation\}\}$

CommandStatusComplete: $CommandStatus(AircraftRoutes(aircraft)) := COMPLETE$

AircraftRouteIDUpdate: $AircraftRouteIDs(aircraft) := NO\_ROUTE$

end

Event ReportLocationHomeRecall ⟨ordinary⟩ $\widehat{=}$

extends ReportLocation

any

$aircraft$

$newlocation$

$timestamp$

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

grd1: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp)$
$\land\ RecentTimestamp(aircraft) < timestamp$

grd2: $aircraft \in dom(AircraftLocationHistory)$

grd10: $aircraft \in dom(AircraftRoutes)$

grd3: $newlocation \in AircraftRoutes(aircraft)$

grd4: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land$
$CommandStatus(AircraftRoutes(aircraft)) \in \{SPECIAL\}$

grd5: $aircraft \in dom(AircraftRouteIDs) \land AircraftRouteIDs(aircraft) \in$
$dom(AircraftRouteTasks)$

grd6: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \in \mathbb{P}(LOCATIONS)$

grd7: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{HOME\} = \varnothing$

grd8: $newlocation \in AircraftRouteTasks(AircraftRouteIDs(aircraft))$

grd9: $NO\_ROUTE \in CurrentIdentifiers$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

act1: $AircraftLocationHistory(aircraft) := (AircraftLocationHistory(aircraft) \cup$
$\{newlocation\})$

act2: $CommandStatus(AircraftRoutes(aircraft)) := SPECIAL$

act3: $AircraftRouteIDs(aircraft) := NO\_ROUTE$

end

Event ReportDeviation ⟨ordinary⟩ $\widehat{=}$

extends ReportDeviation

any

$deviation$

$aircraft$

$timestamp$

where

deviationType: $deviation \in LOCATIONS$

aircraftType: $aircraft \in RegisteredAircraft$

timestampGuard: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp)$
$\land\ RecentTimestamp(aircraft) < timestamp$

aircraftDomainLocation: $aircraft \in dom(AircraftLocationHistory)$

aircraftDomainRoute: $aircraft \in dom(AircraftRoutes)$

deviationNotPartOfRoute: $deviation \notin AircraftRoutes(aircraft)$

deviationNotNoneOrHome: $deviation \neq HOME$

grd1: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land$
$CommandStatus(AircraftRoutes(aircraft)) = CONFIRMED$

grd3: $aircraft \in dom(AircraftLocations)$

grd2: $AircraftLocations(aircraft) \neq HOME$

then

updateLocationDeviation: $AircraftLocations(aircraft) := deviation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

AircraftLocationHistoryUpdate: $AircraftLocationHistory(aircraft) :=$
$(AircraftLocationHistory(aircraft) \cup \{deviation\})$

end

Event ConfirmReceipt ⟨ordinary⟩ $\widehat{=}$

extends ConfirmReceipt

any

route_or_update

aircraft

where

RouteOrUpdateType: $route\_or\_update \in \mathbb{P}(LOCATIONS)$

RouteOrUpdateBelongs: $route\_or\_update \in Commands$

aircraftType: $aircraft \in RegisteredAircraft$

aircraftInRoutesDomain: $aircraft \in dom(AircraftRoutes)$

aircraftHasRoute: $AircraftRoutes(aircraft) \neq \varnothing$

commandStatus: $AircraftRoutes(aircraft) \in dom(CommandStatus)$

aircraftRouteAssigned: $CommandStatus(AircraftRoutes(aircraft)) = ASSIGNED$

route_or_update_valid: $route\_or\_update = AircraftRoutes(aircraft)$

then

updateCommandStatus: $CommandStatus(AircraftRoutes(aircraft)) := CONFIRMED$

end

Event ConfirmCancellation ⟨ordinary⟩ $\widehat{=}$

extends ConfirmCancellation

any

command

result

where

commandType: $command \in \mathbb{P}(LOCATIONS)$

commandInCancelled: $command \in CancelledCommands$

resultsSet: $result = TRUE$

then

skip

end

Event QueryPlanValidationState ⟨ordinary⟩ $\widehat{=}$

extends QueryPlanValidationState

any

plan

result

where

planBelongsToPlans: $plan \in (Plans \setminus \{\varnothing, \{HOME\}\})$

planDomain: $plan \in dom(PlanValidationState)$

resultsSet: $result = PlanValidationState(plan)$

then

skip

end

Event QueryCommandValidationState ⟨ordinary⟩ $\widehat{=}$

extends QueryCommandValidationState

any

command

result

where

commandBelongsToCommands: $command \in (Commands \setminus \{\varnothing, \{HOME\}\})$

commandDomain: $command \in dom(CommandValidationState)$

resultsSet: $result = CommandValidationState(command)$

then

skip

end

END

Appendix F

Drone case study - third refinement of formal model - machine & context

CONTEXT DroneContext4

EXTENDS DroneContext3

SETS

TOKENS

OPERATORS

AIRCRAFTSTATE

CONSTANTS

SUSPECT

NORMAL

AXIOMS

`operatorsFinite`: $finite(OPERATORS)$

`operatorsSize`: $card(OPERATORS) = 2$

`aircraftstateAxm`: $partition(AIRCRAFTSTATE, \{SUSPECT\}, \{NORMAL\})$

END

MACHINE DroneMachine4

REFINES DroneMachine3

SEES DroneContext4

VARIABLES

RegisteredAircraft

RestrictedLocations

Plans

Commands

AircraftLocations

AircraftRoutes

PlanValidationState

CommandValidationState

CancelledCommands

PlanValidationQueue [CR1

CommandValidationQueue [CR3

PlanValidationQueueCount [CR1

CommandValidationQueueCount [CR3

RecentTimestamp

CommandStatus [CR4, CR5

AircraftLocationHistory

RouteIDs

AircraftRouteIDs

AircraftRouteTasks

CurrentIdentifiers

UsedTokens

TokenMapping

CurrentOperators

OperatorMappingPlans

OperatorMappingCommands

AircraftStatus

INVARIANTS

`UsedTokensType`: $UsedTokens \subseteq \mathbb{P}(TOKENS)$

`TokenMappingType`: $TokenMapping \in RegisteredAircraft \pfun \mathbb{P}(TOKENS)$

`CurrentOperatorsType`: $CurrentOperators \subseteq OPERATORS$

`OperatorMappingPlansType`: $OperatorMappingPlans \in CurrentOperators \pfun \mathbb{P}(\mathbb{P}(LOCATIONS))$

`OperatingMappingCommandsType`: $OperatorMappingCommands \in CurrentOperators \pfun \mathbb{P}(\mathbb{P}(LOCATIONS))$

OperatorMappingPlansFinite: $\forall o \cdot o \in dom(OperatorMappingPlans) \Rightarrow finite(OperatorMappingPlans(o))$

OperatorMappingCommandsFinite: $\forall o \cdot o \in dom(OperatorMappingCommands) \Rightarrow finite(OperatorMappingCommands(o))$

OperatorMappingPlanSize: $\forall o, i \cdot o \in CurrentOperators \land o \in dom(OperatorMappingPlans) \land i = OperatorMappingPlans(o) \Rightarrow card(i) \leq 10$

OperatorMappingCommandSize: $\forall o, i \cdot o \in CurrentOperators \land o \in dom(OperatorMappingCommands) \land i = OperatorMappingCommands(o) \Rightarrow card(i) \leq 10$

CommandOwnershipType: $CommandOwnership \in CurrentOperators \pfun \mathbb{P}(\mathbb{P}(LOCATIONS))$

AircraftStatusType: $AircraftStatus \in RegisteredAircraft \pfun AIRCRAFTSTATE$

EVENTS

Initialisation ⟨extended⟩

begin

InitRegisteredAircraft: $RegisteredAircraft := \varnothing$

InitRestrictedLocations: $RestrictedLocations := \varnothing$

InitPlans: $Plans := \{\{HOME\}, \varnothing\}$

Home is always a valid plan.

InitCommands: $Commands := \{\{HOME\}, \varnothing\}$

Home is always a valid command.

InitAircraftLocations: $AircraftLocations := \varnothing$

InitAircraftRoutes: $AircraftRoutes := \varnothing$

InitCommandValidationState: $CommandValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitPlanValidationState: $PlanValidationState := \{\{HOME\} \mapsto VALID, \varnothing \mapsto VALID\}$

InitCancelledCommands: $CancelledCommands := \varnothing$

PlanValidationQueueInit: $PlanValidationQueue := \varnothing$

CommandValidationQueueInit: $CommandValidationQueue := \varnothing$

PlanValidationQueueCountInit: $PlanValidationQueueCount := 0$

CommandValidationQueueCountInit: $CommandValidationQueueCount := 0$

RecentTimestampInit: $RecentTimestamp := \varnothing$

CommandStatusInit: $CommandStatus := \{\{HOME\} \mapsto SPECIAL, \varnothing \mapsto SPECIAL\}$

AircraftLocationHistoryInit: $AircraftLocationHistory := \varnothing$

CurrentIdentifiersInit: $CurrentIdentifiers := \{HOME\_ONLY\_ROUTE, NO\_ROUTE\}$

RouteIdentifierInit: $RouteIDs := \{\{HOME\} \mapsto HOME\_ONLY\_ROUTE, \varnothing \mapsto NO\_ROUTE\}$

AircraftRouteIDsInit: $AircraftRouteIDs := \varnothing$

AircraftRouteTasksInit: $AircraftRouteTasks := \{HOME\_ONLY\_ROUTE \mapsto \{HOME\}, NO\_ROUTE \mapsto \varnothing\}$

TokenMappingInit: $TokenMapping := \varnothing$

UsedTokensInit: $UsedTokens := \varnothing$

CurrentOperatorsInit: $CurrentOperators := \{x \cdot x \in OPERATORS | x\}$

OperatoMappingPlansInit: $OperatorMappingPlans := \{x \cdot x \in OPERATORS | x \mapsto \varnothing\}$

OperatorMappingCommandsInit: $OperatorMappingCommands := \{x \cdot x \in OPERATORS | x \mapsto \varnothing\}$

CommandOwnershipInit: $CommandOwnership := \{x \cdot x \in OPERATORS | x \mapsto \varnothing\}$

AircraftStatusInit: $AircraftStatus := \varnothing$

end

Event RegisterAircraft ⟨ordinary⟩ $\widehat{=}$

extends RegisterAircraft

any

$aircraft$

tokens

where

aircraftType: $aircraft \in DRONES$

aircraftNotDuplicate: $aircraft \notin RegisteredAircraft$

norouteCurrent: $NO\_ROUTE \in CurrentIdentifiers$

tokensType: $tokens \in \mathbb{P}(TOKENS)$

tokensSize: $finite(tokens) \land card(tokens) > 20$

tokensUnique: $\forall tok, existing \cdot tok \in tokens \land existing \in UsedTokens \Rightarrow tok \notin existing$

then

addAircraft: $RegisteredAircraft := RegisteredAircraft \cup \{aircraft\}$

addAircraftLocations: $AircraftLocations := AircraftLocations \cup \{aircraft \mapsto HOME\}$

addAircraftRoute: $AircraftRoutes := AircraftRoutes \cup \{aircraft \mapsto \varnothing\}$

act1: $RecentTimestamp := RecentTimestamp \cup \{aircraft \mapsto 0\}$

AircraftLocationHistoryRegister: $AircraftLocationHistory(aircraft) := \{HOME\}$

AircraftRouteIDsRegister: $AircraftRouteIDs := AircraftRouteIDs \cup \{aircraft \mapsto NO\_ROUTE\}$

mapTokens: $TokenMapping(aircraft) := tokens$

usedTokensAdd: $UsedTokens := UsedTokens \cup \{tokens\}$

aircraftStatusAdd: $AircraftStatus(aircraft) := NORMAL$

end

Event RegisterRestrictedLocation ⟨ordinary⟩ $\widehat{=}$

extends RegisterRestrictedLocation

any

$restricted$

$aircraft\_not\_at\_home$

$aircraft\_at\_home$

$aircraft\_with\_restricted\_waypoints$

where

`restrictedType`: $restricted \in LOCATIONS$

`restrictedNotDuplicate`: $restricted \notin RestrictedLocations$

`restrictedNotHome`: $restricted \neq HOME$

`grd1`: $\{HOME\_ONLY\_ROUTE, NO\_ROUTE\} \subseteq CurrentIdentifiers$

`grd2`: $aircraft\_not\_at\_home = \{y \cdot y \in RegisteredAircraft \wedge y \in dom(AircraftLocations) \wedge AircraftLocations(y) \neq HOME | y\}$

`grd3`: $aircraft\_at\_home = \{y \cdot y \in RegisteredAircraft \wedge y \in dom(AircraftLocations) \wedge AircraftLocations(y) = HOME | y\}$

`grd4`: $aircraft\_with\_restricted\_waypoints = \{y \cdot y \in (aircraft\_at\_home \cup aircraft\_not\_at\_home) \wedge y \in dom(AircraftRoutes) \wedge restricted \in AircraftRoutes(y) | y\}$

then

`addRestricted`: $RestrictedLocations := RestrictedLocations \cup \{restricted\}$

`invalidExistingCommands`:

$CommandValidationState := (\{y \cdot y \in dom(CommandValidationState) \wedge restricted \in y | y\} \lhd CommandValidationState) \cup (\lambda q \cdot q \in dom(CommandValidationState) \wedge restricted \in q \wedge CommandValidationState(q) \neq INVALID | INVALID)$

`replaceAffectedAircraft`:

$AircraftRoutes := (AircraftRoutes \rhd \{y \cdot y \in ran(AircraftRoutes) \wedge restricted \in y | y\}) \cup (\lambda q \cdot q \in dom(AircraftRoutes) \wedge q \in dom(AircraftLocations) \wedge AircraftLocations(q) = HOME \wedge restricted \in AircraftRoutes(q) | \varnothing) \cup (\lambda q \cdot q \in dom(AircraftRoutes) \wedge q \in dom(AircraftLocations) \wedge AircraftLocations(q) \neq HOME \wedge restricted \in AircraftRoutes(q) | \{HOME\})$

`act1`:

$AircraftRouteIDs := (aircraft\_with\_restricted\_waypoints \lhd AircraftRouteIDs) \cup \{y \cdot y \in aircraft\_with\_restricted\_waypoints \cap aircraft\_at\_home | y \mapsto NO\_ROUTE\} \cup \{y \cdot y \in aircraft\_with\_restricted\_waypoints \cap aircraft\_not\_at\_home | y \mapsto HOME\_ONLY\_ROUTE\}$

end

Event TransmitPlan ⟨ordinary⟩ $\widehat{=}$

extends TransmitPlan

any

$plan$

operator

where

locationsType: $plan \in \mathbb{P}(LOCATIONS)$

planNotDuplicate: $plan \notin Plans$

planValidationQueueGuard: $PlanValidationQueueCount \in \mathbb{N}$

grd1: $plan \cap RestrictedLocations = \varnothing$

operatorType: $operator \in CurrentOperators \wedge operator \in dom(OperatorMappingPlans)$

OperatingMappingFinite: $finite(OperatorMappingPlans(operator))$

planNotInOperatorMapping: $plan \notin OperatorMappingPlans(operator)$

planIsFinite: $finite(plan)$

combinedFinite: $finite(OperatorMappingPlans(operator) \cup \{plan\})$

operatorCardinality: $card(OperatorMappingPlans(operator)) < 10$

grd2: $card(OperatorMappingPlans(operator)) + card(\{plan\}) \leq 10$

grd3: $card(\{plan\}) = 1$

then

addPlan: $Plans := Plans \cup \{plan\}$

planValidationState: $PlanValidationState(plan) := UNDETERMINED$

PlanAddQueue: $PlanValidationQueue(plan) := PlanValidationQueueCount$

PlanValidationQueueIncrement: $PlanValidationQueueCount := PlanValidationQueueCount + 1$

operatorJobsAdd: $OperatorMappingPlans(operator) := OperatorMappingPlans(operator) \cup \{plan\}$

end

Event ValidatePlan ⟨ordinary⟩ $\widehat{=}$

extends ValidatePlan

any

$plan$

operator

where

planType: $plan \subseteq LOCATIONS$

planRegistered: $plan \in Plans$

planDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin plan$

grd1: $dom(\{plan\} \lhd PlanValidationQueue) \subseteq \mathbb{P}(LOCATIONS)$

grd2: $0 \in ran(PlanValidationQueue)$

grd3: $PlanValidationQueue^{-1}(0) = plan$

planInRange: $plan \subseteq LOCATIONS$

operatorType: $operator \in dom(OperatorMappingPlans)$

operatorInQuestion: $\exists op \cdot op \in dom(OperatorMappingPlans) \wedge \{plan\} \subseteq OperatorMappingPlans(op) \Rightarrow op = operator$

then

validateAction: $PlanValidationState(plan) := VALID$

act1: $PlanValidationQueue := (\lambda q \cdot q \in dom(\{plan\} \lhd PlanValidationQueue) \land q \in \mathbb{P}(LOCATIONS) \land PlanValidationQueue(q) \in 0..PlanValidationQueueCount | PlanValidationQueue(q) - 1)$

act2: $PlanValidationQueueCount := PlanValidationQueueCount - 1$

operatorJobsRemove: $OperatorMappingPlans(operator) := OperatorMappingPlans(operator) \setminus \{plan\}$

end

Event TransmitCommand ⟨ordinary⟩ $\widehat{=}$

extends TransmitCommand

any

$command$

$identifier$

operator

where

commandType: $command \subseteq LOCATIONS$

commandNotDuplicate: $command \notin Commands$

commandContainsHome: $HOME \in command$

grd1: $CommandValidationQueueCount \in \mathbb{N}$

grd2: $command \cap RestrictedLocations = \varnothing$

identifierType: $identifier \in ROUTEIDENTIFIERS$

identifierIsNew: $identifier \notin CurrentIdentifiers$

commandNotIn: $command \neq \{HOME\} \land command \neq \varnothing$

operatorType: $operator \in CurrentOperators$

operatorDom: $operator \in dom(CommandOwnership)$

commandNotInOperatorSet: $command \notin CommandOwnership(operator)$

operatorDomCommands: $operator \in dom(OperatorMappingCommands)$

operatorMappingCommandsSize: $card(OperatorMappingCommands(operator)) < 10$

commandNotInOPC: $command \notin OperatorMappingCommands(operator)$

then

addCommand: $Commands := Commands \cup \{command\}$

commandValidationState: $CommandValidationState(command) := UNDETERMINED$

act1: $CommandValidationQueue(command) := CommandValidationQueueCount$

act2: $CommandValidationQueueCount := CommandValidationQueueCount + 1$

act3: $CommandStatus(command) := UNASSIGNED$

act4: $RouteIDs := RouteIDs \cup \{command \mapsto identifier\}$

act5: $CurrentIdentifiers := CurrentIdentifiers \cup \{identifier\}$

CommandOwnershipSet: $CommandOwnership(operator) := CommandOwnership(operator) \cup \{command\}$

OperatorMappingCommandsSet: $OperatorMappingCommands(operator) := OperatorMappingCommands(operator) \cup \{command\}$

end

Event ValidateCommand ⟨ordinary⟩ $\widehat{=}$

extends ValidateCommand

any

command

operator

where

commandType: $command \subseteq LOCATIONS$

commandRegistered: $command \in Commands$

commandDoesntContainRestricted: $\forall x \cdot x \in RestrictedLocations \Rightarrow x \notin command$

grd1: $dom(\{command\} \lhd CommandValidationQueue) \subseteq \mathbb{P}(LOCATIONS)$

grd2: $0 \in ran(CommandValidationQueue)$

grd3: $CommandValidationQueue^{-1}(0) = command$

operatorType: $operator \in CurrentOperators$

operatorDom: $operator \in dom(OperatorMappingCommands)$

operatorMappingCommandsFinite: $\exists op \cdot op \in dom(OperatorMappingCommands) \land \{command\} \subseteq OperatorMappingCommands(op) \Rightarrow op = operator$

commandInOPC: $command \in OperatorMappingCommands(operator)$

then

commandValidationStateUpdate: $CommandValidationState(command) := VALID$

act1: $CommandValidationQueue := (\lambda q \cdot q \in dom(\{command\} \lhd CommandValidationQueue) \land q \in \mathbb{P}(LOCATIONS) | CommandValidationQueue(q) - 1)$

act2: $CommandValidationQueueCount := CommandValidationQueueCount - 1$

OperatorMappingCommandsReduce: $OperatorMappingCommands(operator) := OperatorMappingCommands(operator) \setminus \{command\}$

end

Event TaskAircraft ⟨ordinary⟩ $\widehat{=}$

extends TaskAircraft

any

aircraft

command

token

where

aircraftType: $aircraft \in RegisteredAircraft$

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

commandDomainCheck: $command \in dom(CommandValidationState)$

commandCheck: $CommandValidationState(command) = VALID$

aircraftRouteDom: $aircraft \in dom(AircraftRoutes)$

aircraftNoRoute:

$AircraftRoutes(aircraft) = \varnothing \lor$
$(AircraftRoutes(aircraft) \neq \varnothing \land$
$AircraftRoutes(aircraft) \in Commands \land$
$CommandStatus(AircraftRoutes(aircraft)) \in \{COMPLETE, SPECIAL\})$

routeNotSpecial: $CommandStatus(command) = UNASSIGNED$

aircraftRouteIDCheck: $RouteIDs(command) \notin \{NO\_ROUTE,$
$HOME\_ONLY\_ROUTE\}$

noExistingAssignment: $RouteIDs(command) \notin ran(AircraftRouteIDs)$

aircraftMapExists: $aircraft \in dom(TokenMapping)$

tokenType: $token \in TokenMapping(aircraft)$

aircraftDom: $aircraft \in dom(AircraftStatus)$

AircraftStatusCheck: $AircraftStatus(aircraft) = NORMAL$

then

taskAircraftGo: $AircraftRoutes(aircraft) := command$

CommandStatusUpdate: $CommandStatus(command) := ASSIGNED$

act1: $AircraftRouteIDs(aircraft) := RouteIDs(command)$

act2: $AircraftRouteTasks(RouteIDs(command)) := command$

withdrawToken: $TokenMapping(aircraft) := (TokenMapping(aircraft) \setminus \{token\})$

end

Event RetaskAircraft ⟨ordinary⟩ $\widehat{=}$

extends RetaskAircraft

any

command

aircraft

existing_command

token

where

commandType: $command \in Commands$

aircraftType: $aircraft \in RegisteredAircraft$

commandDomainCheck: $command \in dom(CommandValidationState)$

commandCheck: $CommandValidationState(command) = VALID$

aircraftRouteDom: $aircraft \in dom(AircraftRoutes)$

existingCommandCheck: $existing\_command = AircraftRoutes(aircraft)$

existingNotBlank: $existing\_command \neq \varnothing$

existingNotSame: $existing\_command \neq command$

existingNotSameTwo: $command \neq \{HOME\} \land command \neq \varnothing$

existingIsCommand:

$existing\_command \in Commands \land existing\_command \in dom(CommandStatus)$

$\land CommandStatus(existing\_command) \notin \{COMPLETE, SPECIAL\}$

CommandStateCheck: $CommandStatus(command) = UNASSIGNED$

aircraftMapExists: $aircraft \in dom(TokenMapping)$

tokenType: $token \in TokenMapping(aircraft)$

then

retaskAircraftGo: $AircraftRoutes(aircraft) := command$

CommandStatusExistingUpdate: $CommandStatus := (\{command, existing\_command\} \lhd CommandStatus) \cup \{command \mapsto ASSIGNED\} \cup \{existing\_command \mapsto UNASSIGNED\}$

act2: $AircraftRouteIDs(aircraft) := RouteIDs(command)$

act3: $AircraftRouteTasks(RouteIDs(command)) := command$

withdrawToken: $TokenMapping(aircraft) := (TokenMapping(aircraft) \setminus \{token\})$

end

Event RecallAircraft ⟨ordinary⟩ $\widehat{=}$

extends RecallAircraft

any

$aircraft$

where

aircraftType: $aircraft \in RegisteredAircraft$

aircraftTypeHard: $aircraft \in DRONES$

aircraftLocationCheck: $AircraftLocations(aircraft) \neq HOME$

grd1: $\{HOME\} \in dom(RouteIDs)$

then

aircraftRouteChange: $AircraftRoutes(aircraft) := \{HOME\}$

act1: $AircraftRouteIDs(aircraft) := RouteIDs(\{HOME\})$

act2: $AircraftRouteTasks(RouteIDs(\{HOME\})) := \{HOME\}$

end

Event CancelCommand ⟨ordinary⟩ $\widehat{=}$

extends CancelCommand

any

$command$

operator

where

commandType: $command \in Commands \setminus \{\{HOME\}, \varnothing\}$

grd1: $CommandStatus(command) \neq COMPLETE$

grd3: $HOME\_ONLY\_ROUTE \in CurrentIdentifiers$

grd4: $command \in dom(RouteIDs)$

operatorType: $operator \in CurrentOperators$

operatorMembership: $operator \in dom(CommandOwnership)$

commandExists: $command \in Commands \setminus \varnothing$

commandOwned: $command \in CommandOwnership(operator)$

then

removeCommands: $Commands := Commands \setminus \{command\}$

removeCommandValidationState: $CommandValidationState := \{command\} \lhd CommandValidationState$

removeAircraftRoutes: $AircraftRoutes := (\lambda q \cdot q \in dom(AircraftRoutes) \land AircraftRoutes(q) = command | \{HOME\})$

addToCancelled: $CancelledCommands := CancelledCommands \cup \{command\}$

CommandStatusRemove: $CommandStatus := \{command\} \lhd CommandStatus$

AircraftRouteIDsChange: $AircraftRouteIDs := (AircraftRouteIDs \rhd \{RouteIDs(command)\}) \cup (\lambda q \cdot q \in RegisteredAircraft \land q \in dom(AircraftRouteIDs) \land AircraftRouteIDs(q) = RouteIDs(command) | HOME\_ONLY\_ROUTE)$

RouteIDsRemove: $RouteIDs := \{command\} \lhd RouteIDs$

removeOwnership: $CommandOwnership := CommandOwnership \rhd \{\{command\}\}$

end

Event ReportLocationElsewhere ⟨ordinary⟩ $\widehat{=}$

extends ReportLocationElsewhere

any

aircraft

newlocation

timestamp

token

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

grd1: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

grd3: $aircraft \in dom(AircraftLocationHistory)$

grd5: $aircraft \in dom(AircraftRoutes)$

grd4: $newlocation \in AircraftRoutes(aircraft)$

grd6: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land CommandStatus(AircraftRoutes(aircraft)) = CONFIRMED$

grd7: $AircraftRoutes(aircraft) \neq \{HOME\}$

grd8: $aircraft \in dom(AircraftRouteIDs) \land AircraftRouteIDs(aircraft) \in dom(AircraftRouteTasks)$

grd9: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \in \mathbb{P}(LOCATIONS)$

grd10: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{HOME\} \neq \varnothing$

grd11: $newlocation \neq HOME$

grd12: $newlocation \in AircraftRouteTasks(AircraftRouteIDs(aircraft))$

aircraftMapExists: $aircraft \in dom(TokenMapping)$

tokenType: $token \in TokenMapping(aircraft)$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

AircraftLocationHistoryUpdate: $AircraftLocationHistory(aircraft) := (AircraftLocationHistory(aircraft) \cup \{newlocation\})$

AircraftRouteTasksUpdate: $AircraftRouteTasks := AircraftRouteTasks \lhd\!\!\!\!- \{AircraftRouteIDs(aircraft) \mapsto AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{newlocation\}\}$

withdrawToken: $TokenMapping(aircraft) := (TokenMapping(aircraft) \setminus \{token\})$

end

Event ReportLocationHomeNormal ⟨ordinary⟩ $\widehat{=}$

extends ReportLocationHomeNormal

any

$aircraft$

$newlocation$

$timestamp$

token

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

grd1: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

grd2: $aircraft \in dom(AircraftLocationHistory)$

grd3: $aircraft \in dom(AircraftRoutes)$

grd5: $newlocation \in AircraftRoutes(aircraft)$

grd4: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land CommandStatus(AircraftRoutes(aircraft)) \in \{CONFIRMED\}$

grd7: $aircraft \in dom(AircraftRouteIDs) \land AircraftRouteIDs(aircraft) \in dom(AircraftRouteTasks)$

grd6: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \in \mathbb{P}(LOCATIONS)$

grd8: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{HOME\} = \varnothing$

grd9: $newlocation \in AircraftRouteTasks(AircraftRouteIDs(aircraft))$

grd10: $NO\_ROUTE \in CurrentIdentifiers$

aircraftMapExists: $aircraft \in dom(TokenMapping)$

tokenType: $token \in TokenMapping(aircraft)$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

AircraftLocationHistoryUpdate: $AircraftLocationHistory(aircraft) := (AircraftLocationHistory(aircraft) \cup \{newlocation\})$

AircraftRouteTasksUpdate: $AircraftRouteTasks := AircraftRouteTasks \lhd\!\!\!\!- \{AircraftRouteIDs(aircraft) \mapsto AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{newlocation\}\}$

CommandStatusComplete: $CommandStatus(AircraftRoutes(aircraft)) := COMPLETE$

AircraftRouteIDUpdate: $AircraftRouteIDs(aircraft) := NO\_ROUTE$

withdrawToken: $TokenMapping(aircraft) := (TokenMapping(aircraft) \setminus \{token\})$

end

Event ReportLocationHomeRecall ⟨ordinary⟩ $\widehat{=}$

extends ReportLocationHomeRecall

any

$aircraft$

$newlocation$

$timestamp$

token

where

aircraftType: $aircraft \in RegisteredAircraft$

newlocationType: $newlocation \in LOCATIONS$

grd1: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

grd2: $aircraft \in dom(AircraftLocationHistory)$

grd10: $aircraft \in dom(AircraftRoutes)$

grd3: $newlocation \in AircraftRoutes(aircraft)$

grd4: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land CommandStatus(AircraftRoutes(aircraft)) \in \{SPECIAL\}$

grd5: $aircraft \in dom(AircraftRouteIDs) \land AircraftRouteIDs(aircraft) \in dom(AircraftRouteTasks)$

grd6: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \in \mathbb{P}(LOCATIONS)$

grd7: $AircraftRouteTasks(AircraftRouteIDs(aircraft)) \setminus \{HOME\} = \varnothing$

grd8: $newlocation \in AircraftRouteTasks(AircraftRouteIDs(aircraft))$

grd9: $NO\_ROUTE \in CurrentIdentifiers$

aircraftMapExists: $aircraft \in dom(TokenMapping)$

tokenType: $token \in TokenMapping(aircraft)$

then

updateLocationAction: $AircraftLocations(aircraft) := newlocation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

act1: $AircraftLocationHistory(aircraft) := (AircraftLocationHistory(aircraft) \cup \{newlocation\})$

act2: $CommandStatus(AircraftRoutes(aircraft)) := SPECIAL$

act3: $AircraftRouteIDs(aircraft) := NO\_ROUTE$

withdrawToken: $TokenMapping(aircraft) := (TokenMapping(aircraft) \setminus \{token\})$

end

Event ReportDeviation ⟨ordinary⟩ $\widehat{=}$

extends ReportDeviation

any

$deviation$

$aircraft$

$timestamp$

token

where

deviationType: $deviation \in LOCATIONS$

aircraftType: $aircraft \in RegisteredAircraft$

timestampGuard: $timestamp \in \mathbb{N} \land aircraft \in dom(RecentTimestamp) \land RecentTimestamp(aircraft) < timestamp$

aircraftDomainLocation: $aircraft \in dom(AircraftLocationHistory)$

aircraftDomainRoute: $aircraft \in dom(AircraftRoutes)$

deviationNotPartOfRoute: $deviation \notin AircraftRoutes(aircraft)$

deviationNotNoneOrHome: $deviation \neq HOME$

grd1: $AircraftRoutes(aircraft) \in dom(CommandStatus) \land CommandStatus(AircraftRoutes(aircraft)) = CONFIRMED$

grd3: $aircraft \in dom(AircraftLocations)$

grd2: $AircraftLocations(aircraft) \neq HOME$

aircraftMapExists: $aircraft \in dom(TokenMapping)$

tokenType: $token \in TokenMapping(aircraft)$

then

updateLocationDeviation: $AircraftLocations(aircraft) := deviation$

RecentTimestampUpdate: $RecentTimestamp(aircraft) := timestamp$

AircraftLocationHistoryUpdate: $AircraftLocationHistory(aircraft) := (AircraftLocationHistory(aircraft) \cup \{deviation\})$

withdrawToken: $TokenMapping(aircraft) := (TokenMapping(aircraft) \setminus \{token\})$

end

Event ConfirmReceipt ⟨ordinary⟩ $\widehat{=}$

extends ConfirmReceipt

any

$route\_or\_update$

$aircraft$

token

where

RouteOrUpdateType: $route\_or\_update \in \mathbb{P}(LOCATIONS)$

RouteOrUpdateBelongs: $route\_or\_update \in Commands$

aircraftType: $aircraft \in RegisteredAircraft$

aircraftInRoutesDomain: $aircraft \in dom(AircraftRoutes)$

aircraftHasRoute: $AircraftRoutes(aircraft) \neq \varnothing$

commandStatus: $AircraftRoutes(aircraft) \in dom(CommandStatus)$

aircraftRouteAssigned: $CommandStatus(AircraftRoutes(aircraft)) = ASSIGNED$

route_or_update_valid: $route\_or\_update = AircraftRoutes(aircraft)$

aircraftMapExists: $aircraft \in dom(TokenMapping)$

tokenType: $token \in TokenMapping(aircraft)$

then

updateCommandStatus: $CommandStatus(AircraftRoutes(aircraft)) := CONFIRMED$

withdrawToken: $TokenMapping(aircraft) := (TokenMapping(aircraft) \setminus \{token\})$

end

Event ConfirmCancellation ⟨ordinary⟩ $\widehat{=}$

extends ConfirmCancellation

any

$command$

$result$

where

commandType: $command \in \mathbb{P}(LOCATIONS)$

commandInCancelled: $command \in CancelledCommands$

resultsSet: $result = TRUE$

then

skip

end

Event QueryPlanValidationState ⟨ordinary⟩ $\widehat{=}$

extends QueryPlanValidationState

any

plan

result

where

planBelongsToPlans: $plan \in (Plans \setminus \{\varnothing, \{HOME\}\})$

planDomain: $plan \in dom(PlanValidationState)$

resultsSet: $result = PlanValidationState(plan)$

then

skip

end

Event QueryCommandValidationState ⟨ordinary⟩ $\widehat{=}$

extends QueryCommandValidationState

any

command

result

where

commandBelongsToCommands: $command \in (Commands \setminus \{\varnothing, \{HOME\}\})$

commandDomain: $command \in dom(CommandValidationState)$

resultsSet: $result = CommandValidationState(command)$

then

skip

end

Event MarkAircraftSuspect ⟨ordinary⟩ $\widehat{=}$

any

aircraft

where

aircraftType: $aircraft \in RegisteredAircraft$

aircraftStatusDom: $aircraft \in dom(AircraftStatus)$

aircraftNotAlreadySuspect: $AircraftStatus(aircraft) \neq SUSPECT$

then

AircraftStatusAction: $AircraftStatus(aircraft) := SUSPECT$

end

Event MarkAircraftNormal ⟨ordinary⟩ $\widehat{=}$

any

aircraft

where

aircraftType: $aircraft \in RegisteredAircraft$

aircraftStatusDom: $aircraft \in dom(AircraftStatus)$

aircraftAlreadySuspect: $AircraftStatus(aircraft) = SUSPECT$

aircraftHome: $AircraftRouteIDs(aircraft) = NO\_ROUTE$

aircraftHomeAdditional: $AircraftLocations(aircraft) = HOME$

then

AircraftStatusAction: $AircraftStatus(aircraft) := NORMAL$

end

END

References

[1] A Abdulkhaleq and W Stefan. "A-STPA: An Open Tool Support for System-Theoretic Process Analysis". In: (Jan. 2014). URL: http://www.researchgate.net/publication/265508170_A-STPA_An_Open_Tool_Support_for_System-Theoretic_Process_Analysis.

[2] Jean Raymond Abrial et al. "Rodin: An open toolset for modelling and reasoning in Event-B". In: *International Journal on Software Tools for Technology Transfer* 12.6 (Nov. 2010), pp. 447–466. ISSN: 14332779. DOI: 10.1007/s10009-010-0145-y. URL: http://link.springer.com/10.1007/s10009-010-0145-y.

[3] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010. ISBN: 9780521895569. DOI: DOI:10.1017/CBO9781139195881. URL: https://www.cambridge.org/core/books/modeling-in-eventb/F39FF5F1B60F0AA585718B8E6A4F9DD7.

[4] J.-R. Abrial. *The B-book: Assigning Programs to Meanings*. New York, NY, USA: Cambridge University Press, 1996. ISBN: 0-521-49619-5.

[5] Kittisak Sa-Adaem and Yunyong Teng-Amnuay. "Assessing Privacy Protection in Alumni Service". In: *International Journal of Computer and Electrical Engineering* 5.4 (2013), pp. 424–429. ISSN: 17938163. DOI: 10.7763/IJCEE.2013.V5.745. URL: http://www.ijcee.org/index.php?m=content&c=index&a=show&catid=53&id=807.

[6] Cristina Alcaraz and Javier Lopez. "A Security Analysis for Wireless Sensor Mesh Networks in Highly Critical Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.4 (July 2010), pp. 419–428. DOI: 10.1109/TSMCC.2010.2045373. URL: http://ieeexplore.ieee.org/document/5443456/.

[7] Rob Alexander, Richard Hawkins, and Tim Kelly. "Security Assurance Cases: Motivation and the State of the Art". In: (2011), pp. 1–19. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.221.456&rep=rep1&type=pdf.

[8] Nikolaos Alexiou, Stylianos Basagiannis, and Sophia Petridou. "Formal security analysis of near field communication using model checking". In: *Computers & Security* 60 (2016), pp. 1–14. ISSN: 01674048. DOI: 10.1016/j.cose.2016.03.002. URL: http://www.sciencedirect.com/science/article/pii/S0167404816300244.

[9] H. Arabian-Hoseynabadi, H. Oraee, and P. J. Tavner. "Failure Modes and Effects Analysis (FMEA) for wind turbines". In: *International Journal of Electrical Power and Energy Systems* 32.7 (2010), pp. 817–824. ISSN: 01420615. DOI: 10.1016/j.ijepes.2010.01.019. URL: http://www.sciencedirect.com/science/article/pii/S0142061510000281.

[10] Ramachandra Kamath Arbettu et al. "Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers". In: *2016 17th International Telecommunications Network Strategy and Planning Symposium, Networks 2016 - Conference Proceedings*. IEEE, Sept. 2016, pp. 37–44. ISBN: 9781467389914. DOI: 10.1109/NETWKS.2016.7751150. URL: http://ieeexplore.ieee.org/document/7751150/.

[11] Radhakisan Baheti and Helen Gill. "Cyber-physical systems". In: *The impact of control technology* 12 (2011), pp. 161–166.

[12] Federico Baronti et al. "Design and safety verification of a distributed charge equalizer for modular li-ion batteries". In: *IEEE Transactions on Industrial Informatics* 10.2 (May 2014), pp. 1003–1011. ISSN: 15513203. DOI: 10.1109/TII.2014.2299236. URL: http://ieeexplore.ieee.org/document/6708416/.

[13] Earl T. Barr et al. "The oracle problem in software testing: A survey". In: *IEEE Transactions on Software Engineering* 41.5 (May 2015), pp. 507–525. ISSN: 00985589. DOI: 10.1109/TSE.2014.2372785. URL: http://ieeexplore.ieee.org/document/6963470/.

[14] Paul Baybutt. "A critique of the Hazard and Operability (HAZOP) study". In: *Journal of Loss Prevention in the Process Industries* 33 (Jan. 2015), pp. 52–58. ISSN: 09504230. DOI: 10.1016/j.jlp.2014.11.010. URL: https://www.sciencedirect.com/science/article/pii/S0950423014001983.

[15] Patrick Behm et al. "Météor: A Successful Application of B in a Large Project". In: *FM'99 — Formal Methods: World Congress on Formal Methods in the Development of Computing Systems Toulouse, France, September 20–24, 1999 Proceedings, Volume I*. Ed. by Jeannette M. Wing, Jim Woodcock, and Jim Davies. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 369–387. ISBN: 978-3-540-48119-5. DOI: 10.1007/3-540-48119-2{_}22. URL: http://dx.doi.org/10.1007/3-540-48119-2_22.

[16] Ron Bell. "Introduction to IEC 61508". In: *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software - Volume 55*. SCS '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 3–12. ISBN: 1-920-68237-6. URL: http://dl.acm.org/citation.cfm?id=1151816.1151817.

[17] Izak Benbasat, David K Goldstein, and Melissa Mead. "The Case Research Strategy in Studies of Information Systems". In: *MIS Quarterly* 11.3 (1987), pp. 369–386. ISSN: 02767783. DOI: 10.2307/248684. URL: http://www.jstor.org/stable/248684.

[18] Saddek Bensalem et al. "An overview of SAL". In: *Proceedings of the 5th NASA Langley Formal Methods Workshop*. Williamsburg, VA. 2000.

[19] Chris Bogdiukiewicz et al. "Formal Development of Policing Functions for Intelligent Systems". In: *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)* (Oct. 2017), pp. 194–204. DOI: 10.1109/ISSRE.2017.40. URL: http://ieeexplore.ieee.org/document/8109086/.

[20] Hichem Boudali, Pepijn Crouzen, and Marielle Stoelinga. "Dynamic Fault Tree Analysis Using Input/Output Interactive Markov Chains". In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, June 2007, pp. 708–717. ISBN: 0-7695-2855-4. DOI: 10.1109/DSN.2007.37. URL: http://ieeexplore.ieee.org/document/4273022/.

[21] Abdelkader Bouti and Daoud Ait Kadi. "A state-of-the-art review of FMEA/FMECA". In: *International Journal of Reliability, Quality and Safety Engineering* 1.4 (Dec. 1994), pp. 515–543. ISSN: 0218-5393. DOI: 10.1142/S0218539394000362. URL: http://www.worldscientific.com/doi/abs/10.1142/S0218539394000362.

[22] J.B. Bowles. "The new SAE FMECA standard". In: *Reliability and Maintainability Symposium. 1998 Proceedings. International Symposium on Product Quality and Integrity* (1998), pp. 48–53. ISSN: 0149-144X. DOI: 10.1109/RAMS.1998.653561. URL: http://ieeexplore.ieee.org/document/653561/.

[23] F. den Braber et al. "Model-based security analysis in seven steps - A guided tour to the CORAS method". In: *BT Technology Journal* 25.1 (Jan. 2007), pp. 101–117. ISSN: 13583948. DOI: 10.1007/s10550-007-0013-9. URL: http://link.springer.com/10.1007/s10550-007-0013-9.

[24] Michael Butler and Issam Maamria. "Mathematical Extension in Event-B through the Rodin Theory Component". In: June (2010), pp. 1–9.

[25] Michael Butler and Divakar Yadav. "An incremental development of the Mondex system in Event-B". In: *Formal Aspects of Computing* 20.1 (2008), pp. 61–77. DOI: 10.1007/s00165-007-0061-4. URL: https://link.springer.com/article/10.1007%2Fs00165-007-0061-4.

[26] H. Cai et al. "Modelling Safety Monitors of Safety-Critical Railway Systems by Formal Methods". In: *6th IET Conference on Railway Condition Monitoring (RCM 2014)*. Institution of Engineering and Technology, 2014, pp. 2–2. ISBN: 978-1-84919-913-1. DOI: 10.1049/cp.2014.0993. URL: http://digital-library.theiet.org/content/conferences/10.1049/cp.2014.0993.

[27] Alvaro A Cárdenas et al. "Challenges for Securing Cyber Physical Systems". In: *Workshop on Future Directions in Cyber-physical Systems Security* (2009). URL: https://chess.eecs.berkeley.edu/pubs/601.html.

[28] Common Criteria CC. *Common Criteria for Information Technology Security Evaluation, v3.1*. 3.1. Sept. 2012. URL: http://www.commoncriteriaportal.org/.

[29] Marko Čepin and Borut Mavko. "A dynamic fault tree". In: *Reliability Engineering and System Safety* 75.1 (Jan. 2002), pp. 83–91. ISSN: 09518320. DOI: 10.1016/S0951-8320(01)00121-1. URL: https://www.sciencedirect.com/science/article/pii/S0951832001001211.

[30] Civil Aviation Authority. *The Air Navigation Order 2016*. 2016. URL: http://www.legislation.gov.uk/uksi/2016/765/contents/made.

[31] John Colley and Michael Butler. "A Formal, Systematic Approach to STPA using Event-B Refinement and Proof". In: *21th Safety Critical System Symposium* (Feb. 2013). URL: http://eprints.soton.ac.uk/352155/.

[32] International Electrotechnical Commission. *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*. 2010.

[33] International Electrotechnical Commission. *IEC 61511: Functional safety – Safety instrumented systems for the process industry sector*. 2.1. Geneva, 2017, p. 163.

[34] International Electrotechnical Commission. *IEC 61513: Nuclear power plants - Instrumentation and control important to safety - General requirements for systems*. 2.0. Geneva, Switzerland: International Electrotechnical Commission, 2011, p. 205.

[35] International Electrotechnical Commission. *IEC 62443: Security for industrial automation and control systems*. 1.0. Geneva, Switzerland: International Electrotechnical Commission, 2018, p. 338.

[36] F Crawley and B Tyler. *HAZOP: Guide to Best Practice*. Elsevier Science, 2015. ISBN: 9780323394604. URL: https://books.google.co.uk/books?id=xaf0rQEACAAJ.

[37] Anupam Datta et al. "On adversary models and compositional security". In: *IEEE Security & Privacy* 3 (2010), pp. 26–32.

[38] Alexandre David et al. *Uppaal SMC Tutorial*. Tech. rep. 2018, pp. 1–28. URL: http://www.it.uu.se/research/group/darts/papers/texts/uppaal-smc-tutorial.pdf.

[39] United States. Department of Defense. *Mil-Std-1629a: 1980: Procedures for Performing a Failure Mode, Effects and Criticality Analysis*. Department of Defense, 1980.

[40] Mina Deng et al. "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements". In: *Requirements Engineering* 16.1 (Mar. 2011), pp. 3–32. DOI: 10.1007/s00766-010-0115-7. URL: http://link.springer.com/10.1007/s00766-010-0115-7.

[41] Culture Department of Digital Media and Sport. *Developing our capability in cyber security*. Tech. rep. July. Department of Digital, Media, Culture and Sport, 2015. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/496340/ACE-CSR_Brochure_accessible_2015.pdf.

[42] Department of Energy and Climate Change. *Smart Metering Implementation Programme - End to End Technical Architecture*. 2015. URL: https://www.smartenergycodecompany.co.uk/document-download-centre/download-info/technical-architecture-document/.

[43] T Dimitrakos and B Ritchie. "Model based Security Risk Analysis for Web Applications : The CORAS approach". In: *Security* (2002).

[44] Danny Dolev and Andrew C. Yao. "On the Security of Public Key Protocols". In: *IEEE Transactions on Information Theory* 29.2 (Mar. 1983), pp. 198–208. ISSN: 15579654. DOI: 10.1109/TIT.1983.1056650. URL: http://ieeexplore.ieee.org/document/1056650/.

[45] Jordi Dunjó et al. "Hazard and operability (HAZOP) analysis. A literature review". In: *Journal of Hazardous Materials* 173.1-3 (Jan. 2010), pp. 19–32. ISSN: 03043894. DOI: 10.1016/j.jhazmat.2009.08.076. URL: https://www.sciencedirect.com/science/article/pii/S0304389409013727?via%3Dihub.

[46] George B Dyson. *Darwin Among the Machines: The Evolution of Global Intelligence*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN: 0201406497. URL: https://dl.acm.org/citation.cfm?id=523531.

[47] Clifton a. Ericson. "Fault Tree Analysis – A History". In: *The 17th International System Safety Conference* (1999).

[48] European Union Agency for Cybersecurity (ENISA). *Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures*. November. 2017, pp. –. ISBN: 978-92-9204-236-3. DOI: 10.2824/03228. URL: https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot.

[49] C H Fleming, M Seth Placke, and Nancy G Leveson. *Technical Report: STPA Analysis of NextGen Management Components: Ground Interval Management (GIM) and Flight Deck Interval Management (FIM)*. Tech. rep. Cambridge, Massachusetts: Massachusetts Institute of Technology, 2013, p. 95. URL: http://sunnyday.mit.edu/NextGen-Report.pdf.

[50] Rune Fredriksen et al. "The CORAS Framework for a Model-Based Risk Management Process". In: *Safecomp 2002*. 2002. ISBN: 9783540441571. DOI: 10.1007/3-540-45732-1{_}11.

[51] Ivo Friedberg et al. "STPA-SafeSec: Safety and security analysis for cyber-physical systems". In: *Journal of Information Security and Applications* 34 (2017), pp. 183–196. ISSN: 22142126. DOI: 10.1016/j.jisa.2016.05.008. URL: http://www.sciencedirect.com/science/article/pii/S2214212616300850.

[52] Amjad Gawanmeh et al. "Formal Verification of Secrecy in Group Key Protocols Using Event-B". In: *Int. J. Communications, Network and System Sciences* 5.03 (Mar. 2012), pp. 165–177. ISSN: 1913-3715. DOI: 10.4236/ijcns.2012.53021. URL: http://www.scirp.org/journal/doi.aspx?DOI=10.4236/ijcns.2012.53021.

[53] Warren Gilchrist. "Modelling failure modes and effects analysis". In: *International Journal of Quality & Reliability Management* 10.5 (1993). ISSN: 0265671X. DOI: 10.1108/02656719310040105. URL: https://doi.org/10.1108/02656719310040105.

[54] Bill Haskins et al. "Error Cost Escalation Through the Project Life Cycle". In: *INCOSE International Symposium* 14.1 (June 2004), pp. 1723–1737. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.2004.tb00608.x. URL: https://doi.org/10.1002/j.2334-5837.2004.tb00608.x.

[55] Shawn Hernan et al. "Threat modeling-uncover security design flaws using the stride approach". In: *MSDN Magazine* November (Nov. 2006), pp. 68–75. ISSN: 1528-4859.

[56] Thai S. Hoang, David Basin, and Jean-Raymond Abrial. "Specifying Access Control in Event-B". In: *Technical report* 624 (2009). DOI: 10.3929/ETHZ-A-006733720. URL: https://www.research-collection.ethz.ch/handle/20.500.11850/15887.

[57] Thai Son Hoang. "Reasoning about almost-certain convergence properties using Event-B". In: *Science of Computer Programming* 81 (Feb. 2014), pp. 108–121. ISSN: 0167-6423. DOI: 10.1016/J.SCICO.2013.08.006. URL: https://www.sciencedirect.com/science/article/pii/S0167642313001998?via%3Dihub.

[58] Thai Son Hoang et al. "Rodin: an open toolset for modelling and reasoning in Event-B". In: *International Journal on Software Tools for Technology Transfer* 12.6 (2010), pp. 447–466. ISSN: 1433-2779. DOI: 10.1007/s10009-010-0145-y.

[59] C. A. R. Hoare. "Communicating sequential processes". In: *Communications of the ACM*. Vol. 21. 8. Springer, 1978, pp. 666–677. DOI: 10.1145/359576.359585.

[60] Jon Holt and Simon Perry. *SysML for Systems Engineering: A Model-Based Approach*. 2018. ISBN: 9781849196512. DOI: 10.1049/pbpc020e.

[61] Giles Howard et al. "A methodology for assuring the safety and security of critical infrastructure based on STPA and Event-B". In: *International Journal of Critical Computer-Based Systems* 9.1/2 (2019), p. 56. ISSN: 1757-8779. DOI: 10.1504/ijccbs.2019.10020048. URL: http://www.inderscience.com/link.php?id=98815.

[62] G. Howard et al. "Formal analysis of safety and security requirements of critical systems supported by an extended STPA methodology". In: *Proceedings - 2nd IEEE European Symposium on Security and Privacy Workshops, EuroS and PW 2017*. 2017. ISBN: 9780769561073. DOI: 10.1109/EuroSPW.2017.68.

[63] IEC 61882. "Hazard and operability studies (HAZOP studies)-application guide. [IEC 61882]". In: *International Electrotechnical Commission* (2001).

[64] IoT Security Foundation. "IOT Security Archtecture and Policy for the Home - a Hub Based Approach". 2018. URL: https://www.iotsecurityfoundation.org/wp-content/uploads/2018/11/IoT-Security-Architecture-and-Policy-for-the-Home-a-Hub-Based-Approach.pdf.

[65] Takuto Ishimatsu et al. "Hazard analysis of complex spacecraft using systems-theoretic process analysis". In: *Journal of Spacecraft and Rockets* 51.2 (2014), pp. 509–522. DOI: 10.2514/1.A32449. URL: https://arc.aiaa.org/doi/pdf/10.2514/1.A32449.

[66] E N ISO. "13849-1. Safety of machinery, Safety-related parts of control systems, Part 1: General principles for design". In: *International Organization for Standardization* (2015).

[67] Michael Jastram and Michael Butler. *Rodin User's Handbook*. Tech. rep. 2012. URL: https://www3.hhu.de/stups/handbook/rodin/current/pdf/rodin-doc.pdf.

[68] Sohag Kabir. *An overview of fault tree analysis and its application in model based dependability analysis*. July 2017. DOI: 10.1016/j.eswa.2017.01.058. URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417417300714.

[69] Bernhard Kaiser, Catharina Gramlich, and Marc Förster. "State/event fault trees-A safety analysis model for software-controlled systems". In: *Reliability Engineering and System Safety* 92.11 (Nov. 2007), pp. 1521–1537. ISSN: 09518320. DOI: 10.1016/j.ress.2006.10.010. URL: https://www.sciencedirect.com/science/article/pii/S0951832006002092.

[70] Nektarios Karanikas, Maria Mikela Chatzimichailidou, and Martin Rejzek. *International Cross-Industry Safety Conference (ICSC) and European STAMP Workshop and Conference (ESWC) 2018 Editorial of Proceedings*. Tech. rep. 2018, p. 18. URL: https://www.matec-conferences.org/articles/matecconf/pdf/2019/22/matecconf_ICSC-ESWC2018_About-the-Conference.pdf.

[71] Vikash Katta et al. "Requirements management in a combined process for safety and security assessments". In: *Proceedings - 2013 International Conference on Availability, Reliability and Security, ARES 2013*. IEEE, Sept. 2013, pp. 780–786. ISBN: 9780769550084. DOI: 10.1109/ARES.2013.104. URL: http://ieeexplore.ieee.org/document/6657320/.

[72] Rafiullah Khan et al. "STRIDE-based threat modeling for cyber-physical systems". In: *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe, ISGT-Europe 2017 - Proceedings*. Vol. 2018-Janua. IEEE, Sept. 2018, pp. 1–6. ISBN: 9781538619537. DOI: 10.1109/ISGTEurope.2017.8260283. URL: http://ieeexplore.ieee.org/document/8260283/.

[73] Jin Hyun Kim et al. "Formal analysis and testing of real-time automotive systems using UPPAAL tools". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9128. Springer, Cham, 2015, pp. 47–61. ISBN: 9783319194578. DOI: 10.1007/978-3-319-19458-5{_}4. URL: http://link.springer.com/10.1007/978-3-319-19458-5_4.

[74] Rowan Klöti, Vasileios Kotronis, and Paul Smith. "OpenFlow: A security analysis". In: *Proceedings - International Conference on Network Protocols, ICNP*. IEEE, Oct. 2013, pp. 1–6. ISBN: 9781479912704. DOI: 10.1109/ICNP.2013.6733671. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6733671.

[75] Siwar Kriaa et al. *A survey of approaches combining safety and security for industrial control systems*. 2015. DOI: 10.1016/j.ress.2015.02.008. URL: http://dx.doi.org/10.1016/j.ress.2015.02.008.

[76] Sentot Kromodimoeljo and Peter A Lindsay. "Automatic Generation of Minimal Cut Sets". In: *Electronic Proceedings in Theoretical Computer Science* 184 (2015), pp. 33–47. ISSN: 2075-2180. DOI: 10.4204/EPTCS.184.3. URL: https://arxiv.org/pdf/1506.03555.pdf%20http://arxiv.org/abs/1506.03555.

[77] Rajesh Kumar and Marielle Stoelinga. "Quantitative security and safety analysis with attack-fault trees". In: *Proceedings of IEEE International Symposium on High Assurance Systems Engineering* (2017), pp. 25–32. ISSN: 15302059. DOI: 10.1109/HASE.2017.12.

[78] Robert Künnemann and Graham Steel. "YubiSecure? Formal Security Analysis Results for the Yubikey and YubiHSM". In: *Security and Trust Management: 8th International Workshop, STM 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*. Ed. by Audun Jøsang, Pierangela Samarati, and Marinella Petrocchi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 257–272. ISBN: 978-3-642-38004-4. DOI: 10.1007/978-3-642-38004-4{_}17. URL: http://dx.doi.org/10.1007/978-3-642-38004-4_17.

[79] Kim G. Larsen, Marius Mikucionis, and Brian Nielsen. "Online Testing of Real-time Systems Using Uppaal". In: *Formal Approaches to Software Testing*. Ed. by Jens Grabowski and Brian Nielsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 79–94. ISBN: 978-3-540-31848-4. DOI: 10.1007/978-3-540-31848-4{_}6. URL: http://link.springer.com/10.1007/978-3-540-31848-4_6.

[80] Kim G Larsen, Paul Pettersson, and Wang Yi. *Uppaal in a nutshell*. 1997. DOI: 10.1007/s100090050010. URL: http://user.it.uu.se/~yi/pdf-files/2017/nutshell.pdf.

[81] W. S. Lee et al. "Fault Tree Analysis, Methods, and Applications - A Review". In: *IEEE Transactions on Reliability* R-34.3 (Aug. 1985), pp. 194–203. ISSN: 15581721. DOI: 10.1109/TR.1985.5222114. URL: http://ieeexplore.ieee.org/document/5222114/.

[82] W. S. Lee et al. "Fault Tree Analysis, Methods, and Applications - A Review". In: *IEEE Transactions on Reliability* R-34.3 (Aug. 1985), pp. 194–203. ISSN: 15581721. DOI: 10.1109/TR.1985.5222114. URL: http://ieeexplore.ieee.org/document/5222114/.

[83] Michael Leuschel and Michael Butler. "ProB: A Model Checker for B". In: *FME 2003 Formal Methods* (2003). ISSN: 03029743. DOI: 10.1007/b13229.

[84] Michael Leuschel and Michael Butler. "ProB: An automated analysis toolset for the B method". In: *International Journal on Software Tools for Technology Transfer* (2008). ISSN: 14332779. DOI: 10.1007/s10009-007-0063-9.

[85] Nancy Leveson. "A new accident model for engineering safer systems". In: *Safety Science* 42.4 (Apr. 2004), pp. 237–270. ISSN: 09257535. DOI: 10.1016/S0925-7535(03)00047-X. URL: http://www.sciencedirect.com/science/article/pii/S092575350300047X.

[86] Nancy Leveson. "An STPA Primer". Cambridge, Massachusetts, 2013. URL: https://psas.scripts.mit.edu/home/home/stpa-primer/.

[87] Nancy Leveson. *STAMP Workshop - Partnership for Systems Approaches to Safety and Security*. 2019.

[88] Nancy G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press, 2011, p. 555. ISBN: 9780262016629. DOI: 10.1017/CBO9781107415324.004. URL: https://mitpress.mit.edu/books/engineering-safer-world.

[89] Nancy G Leveson. "Rasmussen's legacy: A paradigm change in engineering for safety". In: *Applied Ergonomics* 59.Part B (2017), pp. 581–591. ISSN: 18729126. DOI: 10.1016/j.apergo.2016.01.015. URL: http://www.sciencedirect.com/science/article/pii/S0003687016300151.

[90] Nancy G Leveson, Margaret] Stringfellow, and John Thomas. *A Systems Approach to Accident Analysis*. Tech. rep. Cambridge, Massachusetts: Massachusetts Institute of Technology, 2009. URL: http://sunnyday.mit.edu/safer-world/refinery-edited.doc.

[91] Nancy Leveson and John Thomas. *STPA Handbook*. 2018, p. 188. URL: http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf.

[92] Nancy Leveson et al. "Engineering Resilience into Safety-Critical Systems". In: *Resilience Engineering: Concepts And Precepts* (2006), pp. 95–123. ISSN: 0-7546-4641-6.

[93] Ian Levy and National Cyber Security Centre. *The smart security behind the GB Smart Metering System*. 2016. URL: https://www.ncsc.gov.uk/information/the-smart-security-behind-the-gb-smart-metering-system.

[94] Elena Lisova, Irfan Sljivo, and Aida Causevic. "Safety and Security Co-Analyses: A Systematic Literature Review". In: *IEEE Systems Journal* PP (2018), pp. 1–12. ISSN: 1932-8184. DOI: 10.1109/JSYST.2018.2881017. URL: https://ieeexplore.ieee.org/document/8556001/.

[95] lucidchart. *Online Diagram Software and Visual Solution — Lucidchart*. 2017. URL: https://www.lucidchart.com/.

[96] Derek Mannering, Jon G Hall, and Lucia Rapanotti. "Safety process improvement with POSE and alloy". In: *Improvements in System Safety - Proceedings of the 16th Safety-Critical Systems Symposium, SSS 2008*. 2008, pp. 25–41. ISBN: 9781848000995. DOI: 10.1007/978-1-84800-100-8{_}3. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-540-75101-4_23.pdf.

[97] Sjouke Mauw and Martijn Oostdijk. "Foundations of Attack Trees". In: *Information Security and Cryptology - ICISC 2005*. Ed. by Dong Ho Won and Seungjoo Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 186–198. ISBN: 978-3-540-33355-5.

[98] John McDermott and Chris Fox. "Using abuse case models for security requirements analysis". In: *Proceedings - Annual Computer Security Applications Conference, ACSAC*. Vol. Part F1334. IEEE Comput. Soc, 1999, pp. 55–64. ISBN: 0769503462. DOI: 10.1109/CSAC.1999.816013. URL: http://ieeexplore.ieee.org/document/816013/.

[99] Thomas C. McKelvey. "How to Improve the Effectiveness of Hazard and Operability Analysis". In: *IEEE Transactions on Reliability* 37.2 (June 1988), pp. 167–170. ISSN: 15581721. DOI: 10.1109/24.3737. URL: http://ieeexplore.ieee.org/document/3737/.

[100] Murat Moran, James Heather, and Steve Schneider. "Verifying anonymity in voting systems using CSP". In: *Formal Aspects of Computing* 26.1 (Jan. 2014), pp. 63–98. ISSN: 09345043. DOI: 10.1007/s00165-012-0268-x. URL: http://link.springer.com/10.1007/s00165-012-0268-x.

[101] Shoichi Morimoto et al. "Formal verification of security specifications with common criteria". In: *Proceedings of the 2007 ACM Symposium on Applied computing - SAC '07*. New York, New York, USA: ACM Press, 2007, pp. 1506–1512. ISBN: 1595934804. DOI: 10.1145/1244002.1244325. URL: http://portal.acm.org/citation.cfm?doid=1244002.1244325.

[102] Theodore Mouroutis and Athanasios Lioumpas. *RERUM Deliverable D2.1 Use-cases definition and threat analysis*. Tech. rep. 2014. URL: https://bscw.ict-rerum.eu/pub/bscw.cgi/d14540/RERUM_deliverable_D2_1_rev1_1.pdf.

[103] Chunyan Mu. "On information flow control in event-B and refinement". In: *Proceedings - 2013 International Symposium on Theoretical Aspects of Software Engineering, TASE 2013*. IEEE, July 2013, pp. 225–232. ISBN: 9780768550534. DOI: 10.1109/TASE.2013.43. URL: http://ieeexplore.ieee.org/document/6597902/.

[104] Jan Peter Nicklas et al. "Use case based approach for an integrated consideration of safety and security aspects for smart home applications". In: *2016 11th Systems of Systems Engineering Conference, SoSE 2016*. IEEE, June 2016, pp. 1–6. ISBN: 9781467387279. DOI: 10.1109/SYSOSE.2016.7542908. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7542908.

[105] Tope Omitola, Abdolbaghi Rezazadeh, and Michael Butler. "Making (Implicit) Security Requirements Explicit for Cyber-Physical Systems: A Maritime Use Case Security Analysis". In: *Database and Expert Systems Applications*. Ed. by Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil. Cham: Springer International Publishing, Aug. 2019, pp. 75–84. ISBN: 978-3-030-27684-3. DOI: 10.1007/978-3-030-27684-3{_}11. URL: http://link.springer.com/10.1007/978-3-030-27684-3_11.

[106] R.S.H. Piggin. "Development of industrial cyber security standards: IEC 62443 for SCADA and industrial control system security". In: *IET Conference on Control and Automation 2013: Uniting Problems and Solutions*. Institution of Engineering and Technology, 2013, pp. 11–11. ISBN: 978-1-84919-710-6. DOI: 10.1049/cp.2013.0001. URL: https://digital-library.theiet.org/content/conferences/10.1049/cp.2013.0001.

[107] Sam Procter. "A development and assurance process for Medical Application Platform apps". PhD thesis. Kansas State University, 2016, p. 276. URL: http://www.clausewitz.com/bibl/Kipp-MilitarizationOfMarxism.pdf.

[108] Javier Puente et al. "A decision support system for applying failure mode and effects analysis". In: *International Journal of Quality & Reliability Management* 19.2 (2002), pp. 137–150. ISSN: 0265-671X. DOI: 10.1108/02656710210413480. URL: https://www.emeraldinsight.com/doi/pdfplus/10.1108/02656710210413480.

[109] Nafees Qamar, Yves Ledru, and Akram Idani. "Validation of security-design models using Z". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6991 LNCS. Springer, Berlin, Heidelberg, Oct. 2011, pp. 259–274. ISBN: 9783642245589. DOI: 10.1007/978-3-642-24559-6{_}19. URL: http://link.springer.com/10.1007/978-3-642-24559-6_19.

[110] Jens Rasmussen. "Risk management in a dynamic society: a modelling problem". In: *Safety Science* 27.2-3 (Nov. 1997), pp. 183–213. ISSN: 09257535. DOI: 10.1016/S0925-7535(97)00052-0. URL: http://www.sciencedirect.com/science/article/pii/S0925753597000520.

[111] Christian Raspotnig, Peter Karpati, and Vikash Katta. "A Combined Process for Elicitation and Analysis of Safety and Security Requirements". In: *Enterprise, Business-Process and Information Systems Modeling*. Ed. by Ilia Bider et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 347–361. ISBN: 978-3-642-31072-0.

[112] Christian Raspotnig et al. "Enhancing CHASSIS: A method for combining safety and security". In: *Proceedings - 2013 International Conference on Availability, Reliability and Security, ARES 2013*. IEEE, Sept. 2013, pp. 766–773. ISBN: 9780769550084. DOI: 10.1109/ARES.2013.102. URL: http://ieeexplore.ieee.org/document/6657318/.

[113] Felix Redmill, Morris Chudleigh, and James Catmur. *System Safety: HAZOP and Software HAZOP*. 2000. ISBN: 0471982806 9780471982807. DOI: 10.1108/imds.2000.100.1.46.2.

[114] Abdolbaghi Rezazadeh et al. "Redevelopment of an Industrial Case Study Using Event-B and Rodin". In: *Proceedings of the 2007th internatioanal conference on Formal Methods in Industry (FACS-FMI'07)* (2007), pp. 1–8.

[115] Ken Robinson. *System Modelling & Design Using Event-B*. Sydney: The University of New South Wales, 2010, p. 136. URL: http://wiki.event-b.org/images/SM%26D-KAR.pdf.

[116] Ron Ross, Michael McEvilley, and Janet Carrier Oren. "Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems". In: *NIST Special Publication* (Nov. 2016), pp. 800–160. DOI: 10.6028/NIST.SP.800-160. URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf.

[117] Enno Ruijters and Mariëlle Stoelinga. *Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools*. Feb. 2015. DOI: 10.1016/j.cosrev.2015.03.001. URL: https://www.sciencedirect.com/science/article/pii/S1574013715000027.

[118] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual*. 2004. ISBN: 0321245628.

[119] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (2008), p. 131. ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8. URL: https://doi.org/10.1007/s10664-008-9102-8.

[120] Neil Salkind. *Encyclopedia of Research Design*. Thousand Oaks, California, 2010. DOI: 10.4135/9781412961288. URL: https://methods.sagepub.com/reference/encyc-of-research-design.

[121] Christoph Schmittner, Zhendong Ma, and Paul Smith. "FMVEA for safety and security analysis of intelligent and cooperative vehicles". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8696 LNCS. 2014, pp. 282–288. ISBN: 9783319105567. DOI: 10.1007/978-3-319-10557-4{_}31. URL: http://link.springer.com/10.1007/978-3-319-10557-4_31.

[122] Christoph Schmittner et al. "A Case Study of FMVEA and CHASSIS as Safety and Security Co-Analysis Method for Automotive Cyber-physical Systems". In: *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security - CPSS '15*. CPSS '15. New York, NY, USA: ACM, 2015, pp. 69–80. ISBN: 9781450334488. DOI: 10.1145/2732198.2732204. URL: http://doi.acm.org/10.1145/2732198.2732204.

[123] Christoph Schmittner et al. "Security application of Failure Mode and Effect Analysis (FMEA)". In: *33rd International Conference, SAFECOMP 2014*. Springer International Publishing, 2014, pp. 310–325. ISBN: 9783319105055. DOI: 10.1007/978-3-319-10506-2{_}21. URL: http://www.arrowhead.eu/wp-content/uploads/2013/03/FMVEA_camera_ready.pdf.

[124] S. Schneider. "Security properties and CSP". In: *Proceedings of the 1996 IEEE Conference on Security and Privacy*. SP'96. Washington, DC, USA: IEEE Computer Society, 2002, pp. 174–187. ISBN: 0-8186-7417-2. DOI: 10.1109/secpri.1996.502680. URL: http://dl.acm.org/citation.cfm?id=1947337.1947362.

[125] Steve Schneider. "Verifying authentication protocols with CSP". In: *Proceedings - IEEE Computer Security Foundations Symposium*. Vol. 24. 9. 1997, pp. 3–17. ISBN: 0818679905. DOI: 10.1109/CSFW.1997.596775. URL: http://ieeexplore.ieee.org/document/713329/.

[126] Bruce Schneier. "Attack Trees". In: *Dr. Dobb's Journal of Software Tools* 24.12 (1999), p. 60. ISSN: 1044-789X. URL: http://www.schneier.com/paper-attacktrees-ddj-ft.html.

[127] Darren Seifert and Hassan Reza. "A Security Analysis of Cyber-Physical Systems Architecture for Healthcare". In: *Computers* 5.4 (Oct. 2016), p. 27. ISSN: 2073-431X. DOI: 10.3390/computers5040027. URL: http://www.mdpi.com/2073-431X/5/4/27.

[128] Koushik Sen, Mahesh Viswanathan, and Gul Agha. "Statistical Model Checking of Black-Box Probabilistic Systems". In: Springer, Berlin, Heidelberg, 2010, pp. 202–215. DOI: 10.1007/978-3-540-27813-9{_}16. URL: http://link.springer.com/10.1007/978-3-540-27813-9_16.

[129] Stuart S Shapiro. "Privacy Risk Analysis Based on System Control Structures: Adapting System-Theoretic Process Analysis for Privacy Engineering". In: *Proceedings - 2016 IEEE Symposium on Security and Privacy Workshops, SPW 2016*. 2016, pp. 17–24. ISBN: 9781509008247. DOI: 10.1109/SPW.2016.15. URL: http://ieeexplore.ieee.org/document/7527748/.

[130] Renato Silva et al. "Decomposition Tool for Event-B". In: *Software: Practice and Experience* 41.2 (2011), pp. 199–208.

[131] Guttorm Sindre and Andreas L Opdahl. "Eliciting security requirements with misuse cases". In: *Requirements engineering* 10.1 (2005), pp. 34–44. DOI: 10.1007/s00766-004-0194-4. URL: https://link.springer.com/article/10.1007%2Fs00766-004-0194-4.

[132] Doaa Soliman, Kleanthis Thramboulidis, and Georg Frey. "Transformation of Function Block Diagrams to UPPAAL timed automata for the verification of safety applications". In: *Annual Reviews in Control* 36.2 (Dec. 2012), pp. 338–345. ISSN: 13675788. DOI: 10.1016/j.arcontrol.2012.09.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S1367578812000508.

[133] University Of Southampton. *Cyber Physical Systems Research Group*. 2019. URL: https://www.cps.ecs.soton.ac.uk.

[134] University of Southampton. *Cyber Security Research Group*. 2019. URL: https://cyber.southampton.ac.uk/.

[135] J Spivey. *The Z notation: A reference manual*. Second Edi. Hemel Hempstead: Prentice Hall, 1992. ISBN: 0-13-978529-9. DOI: 10.1016/0167-6423(90)90091-Q. URL: http://spivey.oriel.ox.ac.uk/mike/zrm/.

[136] Christian Spreafico, Davide Russo, and Caterina Rizzi. *A state-of-the-art review of FMEA/FMECA including patents*. Aug. 2017. DOI: 10.1016/j.cosrev.2017.05.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1574013716301435.

[137] Yannis C. Stamatiou et al. "The CORAS approach for model-based risk management applied to a telemedicine service". In: *Studies in Health Technology and Informatics*. 2003. ISBN: 1586033476. DOI: 10.3233/978-1-60750-939-4-206.

[138] Stamatis, D.H. "Failure Mode and Effect Analysis: FMEA from Theory to Execution". In: *American Society For Quality, Quality Press, Milwaukee* (2003). ISSN: 00401706. DOI: 10.2307/1268911.

[139] Andreas Svendsen, Øystein Haugen, and Birger Møller-Pedersen. "Specifying a testing oracle for train stations - Going beyond with product line technology". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7167 LNCS. Springer, Berlin, Heidelberg, Oct. 2012, pp. 187–201. ISBN: 9783642296444. DOI: 10.1007/978-3-642-29645-1{_}20. URL: http://link.springer.com/10.1007/978-3-642-29645-1_20.

[140] Hideo Tanaka et al. "FAULT-TREE ANALYSIS BY FUZZY PROBABILITY". In: *IEEE Transactions on Reliability* R-32.5 (Dec. 1983), pp. 453–457. ISSN: 00189529. DOI: 10.1109/TR.1983.5221727. URL: http://ieeexplore.ieee.org/document/5221727/.

[141] John Thomas. "Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis". PhD thesis. Massachusetts Institute of Technology, 2013, p. 232. DOI: 10.2172/1044959. URL: https://dspace.mit.edu/handle/1721.1/81055.

[142] Erik Nilsen Torkildson et al. "Empirical studies of methods for safety and security co-analysis of autonomous boat". In: *Safety and Reliability – Safe Societies in a Changing World*. 2018, pp. 2949–2957. ISBN: 9780815386827. DOI: 10.1201/9781351174664-369. URL: www.dnvgl.com/technology-innovation/.

[143] Jørn Vatn. "Finding minimal cut sets in a fault tree". In: *Reliability Engineering & System Safety* 36.1 (Jan. 1992), pp. 59–62. ISSN: 09518320. DOI: 10.1016/0951-8320(92)90152-B. URL: https://www.sciencedirect.com/science/article/pii/095183209290152B.

[144] Venkat Venkatasubramanian, Jinsong Zhao, and Shankar Viswanathan. "Intelligent systems for HAZOP analysis of complex process plants". In: *Computers and Chemical Engineering* 24.9-10 (Oct. 2000), pp. 2291–2302. ISSN: 00981354. DOI: 10.1016/S0098-1354(00)00573-1. URL: https://www.sciencedirect.com/science/article/pii/S0098135400005731.

[145] William Vesely et al. *Fault tree handbook with aerospace applications version 1.1*. Tech. rep. WASHINGTON DC: National Aeronautics and Space Administration, 2002, p. 218. URL: https://elibrary.gsfc.nasa.gov/_assets/doclibBidder/tech_docs/25.%20NASA_Fault_Tree_Handbook_with_Aerospace_Applications%20-%20Copy.pdf.

[146] Jim Woodcock et al. "Formal Methods: Practice and experience". In: *ACM Computing Surveys* 41.4 (Oct. 2009), pp. 1–36. ISSN: 03600300. DOI: 10.1145/1592434.1592436. URL: http://portal.acm.org/citation.cfm?doid=1592434.1592436.

[147] William Young and Nancy G. Leveson. "An integrated approach to safety and security based on systems theory". In: *Communications of the ACM* 57.2 (Feb. 2014), pp. 31–35. ISSN: 00010782. DOI: 10.1145/2556938. URL: http://dl.acm.org/ft_gateway.cfm?id=2556938&type=html.

[148] T. Yuge and S. Yanagi. "Quantitative analysis of a fault tree with priority AND gates". In: *Reliability Engineering and System Safety* 93.11 (Nov. 2008), pp. 1577–1583. ISSN: 09518320. DOI: 10.1016/j.ress.2008.02.016. URL: https://www.sciencedirect.com/science/article/pii/S0951832008000409.

[149] Saad Zafar and R. G. Dromey. "Integrating safety and security requirements into design of an embedded system". In: *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*. Vol. 2005. IEEE, 2005, pp. 629–636. ISBN: 0769524656. DOI: 10.1109/APSEC.2005.75. URL: http://ieeexplore.ieee.org/document/1607203/.