# A formal methodology, based on STPA and leveraging Event-B, for assuring the safety and security of critical infrastructure

**Abstract:** Cyber-physical systems represent a challenge to conventional security and safety analysis techniques due to their complexity and the need to consider both safety and security equally. It is also important that the requirements generated to mitigate against safety and security risks are clear and adequately address the underlying issue. A methodology is presented in this paper to allow for integrated safety and security analysis of cyber-physical systems, particularly in a critical infrastructure context. This methodology uses a modified form of STPA, which has been coupled with our concept of adversarial modelling, to analyse for security and safety hazards which are then mitigated against by the creation of critical requirements. These critical requirements are then validated through their application to an Event-B formal model, allowing for their completeness to be verified. The output of the methodology is a set of critical requirements that guide iteration of and improvements to the system design to ensure its safety and security are maintained.

**Keywords:** Systems-Theoretic Process Analysis; hazard analysis; Event-B; formal methods; safety analysis; security analysis; critical infrastructure; integrated methodology; cyber-physical systems; adversarial modelling; system design; system engineering; formal modelling; requirements engineering; critical systems.

## 1 Introduction

With the increasing use of cyber-physical systems in a critical infrastructure context, a need has arisen to ensure that these systems are inherently secure and safe in their design due to the significant risks of harm that are posed by such systems being undermined by motivated attackers. There are a raft of methodologies that allow for the analysis of safety and security risks in the design of systems, but many of these methodologies only consider safety or security separately, and do not utilise concepts that can be easily understood in other contexts. This can result in ambiguity in the design of the system or - worst still - conflicting safety and security requirements that have not been reconciled adequately. For systems in such a critical context, this can introduce unacceptable compromises, which may ultimately result in an increased safety or security risk.

Our approach in addressing this lack of an integrated methodology covering both security and safety and using unified concepts and processes is therefore centered around the creation of a new methodology that is designed both for use in the context of highly-critical national infrastructure systems, and which allows for the analysis of security and safety issues to be undertaken in a harmonised manner.

We developed our methodology by modifying the existing Systems-Theoretic Process Analysis technique (N. G. Leveson, 2011) and adding a security analysis process to it, as well as providing higher assurance that the artefacts of the analysis meaningfully mitigate against

identified hazardous and adversarial behaviours through the use of a formal model. The approach described in this paper has been developed directly from existing work (Howard et al., 2017) in order to bolster the security aspect of the analysis - this is a result of a detailed exploration of the standard features and concepts of security analysis methodologies within the literature - this aspect receives a detailed exploration in subsection 3.2.

Our aim is therefore to provide a highly traceable approach to performing security and safety analysis in one methodology, utilising unified concepts and terminology to bring together security and safety meaningfully and robustly.

## 2 Background & related work

In this section, we consider three relevant areas of literature to our work - existing safety/hazard analysis methodologies, security/privacy analysis methodologies and finally the use of formal methods in system analysis methodologies. We then discuss some related work that attempts to broadly address the same issues we've identified.

Safety analysis techniques such as HAZOP (Dunjó et al., 2010), Fault Tree Analysis (Lee et al., 1985) and Failure Modes and Effect Analysis (Dhillon, 1992) all originate from the middle of the last century and were designed to deal with the highly analogue systems of the time. However, as systems have become more digital and the focus has shifted from a large number of analogue components to systems being comprised of more complicated and dynamic components with a huge spectrum of possible interactions, these methodologies have struggled to identify an increasing number of hazards and dangerous interactions between components within a system (N. G. Leveson, 2011). This has led to the creation of new methodologies, such as STAMP and its associated analysis technique of STPA (ibid.) which seek to take a more systems theory-based approach to analysing systems for hazards. This approach is rooted in work by Rasmussen, a human factors expert, who understood that the migration of a system from safe behaviour to unsafe behaviour was due to a combination of factors, including human factors issues, more than a linear sequence of 'errors' (Rasmussen, 1997). The primary output of an STPA analysis is therefore focused on the generation of *constraints* either on system behaviours or that of its human operators to ensure that mitigation is built not only into the design of systems, but equally into training, maintenance, etc. around systems such that they do not 'drift' into unsafe behaviours.

Security and privacy analysis also has a selection of useful techniques and approaches to security such as CORAS (Braber et al., 2007) and LINDDUN (Deng et al., 2011) which are designed to take a much more top-down view on system security, and which exist in complement to more specific techniques such as formal protocol analysis which are still highly prevalent in the literature (i.e. analysis of Yubikey security tokens (Künnemann and Steel, 2013)). Techniques have been developed such as attack trees which seek to identify and mitigate all possible attacks on a given system through combinations of both technical and human-based attack paths (Schneier, 1999). This approach resembles fault tree analysis in its usage of a hierarchical set of nodes which may have boolean relationships between each other.

An addition area of consideration for our work is formal methods/formal modelling. The use of formal methods can benefit both security and safety analyses as they represent "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..." (Cai et

al., 2014). Formal methods have been used in order to develop systems from parts of Amazon Web Services (Newcombe et al., 2015) to assurance of the software powering safety-critical aspects of a line of the Paris Metro system (Behm et al., 1999). We pay particular attention to a development of the B-method, known as Event-B (Abrial, 2010), which has been leveraged for both security analysis (Gawanmeh et al., 2012) as well as safety analysis (Rezazadeh et al., 2007) and has therefore proven itself useful in both domains. Furthermore, Event-B has been paired with an STPA methodology in the past which has demonstrated synergies between STPA's constraint-based approach and Event-B's modelling techniques (Colley and Butler, 2013).

The concept of an integrated security and safety analysis is by no means novel within the literature and attempts have been made with methodologies such as CHASSIS (Raspotnig et al., 2013) and attack-fault trees (Kumar and Stoelinga, 2017) - these methodologies seek to treat security and safety as first class citizens, but lack the formalism that we are endeavouring to provide. Work has also been undertaken such as STPA-Sec (Young and N. G. Leveson, 2014), STPA-SafeSec (Friedberg et al., 2017) and *SAFE and Secure* (Procter, Vasserman, and Hatcliff, 2017) which are all in some way developments or inspired by STPA and attempt to introduce security as a first-class citizen into an STPA-style analysis with varying degrees of completeness. Many of these methodologies are not specifically targeted at the national infrastructure context and once again lack the formal analysis capability that our methodology seeks to provide. Finally, STPA has also been extended with formalisms in the past in order to enable automated generation of model-based requirement specifications (Thomas, 2013).

It is therefore clear from the literature that there is a research gap in terms of a fully integrated security and safety analysis methodology, which leverages formal methods in order to provide stronger *traceability* of the generated constraints through formal proof. As we are seeking to design a methodology which is particularly applicable to critical systems, it is important that the traceability of the any design changes/decisions and constraints are backed up as substantially as possible, and the usage of formal methods aid substantially in this regard.

## 3 Development of our methodology

### *3.1 'Standard STPA' compared to our methodology*

In STAMP - Systems-Theoretic Accident Model and Processes - the understanding of systems is based primarily on a view that accidents are due to any combination of 'external disturbances, component failures, or dysfunctional interactions among system components that are not adequately handled by the control system ... they result from inadequate control or enforcement of safety-related constraints on the development, design, and operation of the system' (N. Leveson, 2004). It is on top of this model of accidents that the Systems-Theoretic Process Analysis technique is designed, and consists of two primary steps:

1. Identify the potential for inadequate control of the system that could lead to a hazardous state, through the analysis of all control actions identified within the system.

2. Determine how each potentially hazardous control action identified in Step 1 could occur. (N. G. Leveson, 2011).

The primary 'output' of the STPA analysis is therefore a set of *constraints* that are to be taken forward to either further improve upon the system design and build-in as many of the constraints into the system design as possible or alternatively for the constraints to be taken forward to guide implementation, training and other aspects of the system during its useful life.

This model and analysis process works well for purely safety-based concerns but it can find itself wanting when it is directly applied to security issues, such as in STPA-Sec (Young and N. G. Leveson, 2014), as the STAMP model does not consider - at its core - the risks to a system based on motivated adversaries with intent. In addition, some security issues that have no bearing on system safety can be missed in a standard STPA analysis, such as concerns about confidentiality of information between controllers.

We therefore seek to augment the standard STPA approach with an explicit model of adversarial behaviours in order to ensure that the security risks of malicious control are considered alongside the risks posed to system safety by hazardous control. By doing so in a way that leverages unified concepts, the ultimate output of our process is still *constraints* - which we term *critical requirements* - though these should meaningfully mitigate against both malicious behaviours and hazardous control.

Our methodology therefore differs from STPA in the following substantive ways:

- The addition of adversarial modelling to the analysis to consider explicit security risks to the system from a range of sources.
- Modifications to terminology to unify some elements of the analysis together, as well as the renaming of 'constraints' to 'critical requirements'.
- The addition of the Event-B formal method to help verify the completeness of critical requirements in mitigating against the hazards that generated them, and to guide improvement of the critical requirements such that they do fully mitigate against identified hazards. We believe this to be a more complete method of producing useful critical requirements than simply utilising subject-matter experts (SMEs) as is often recommended in STPA documentation.
- More explicit traceability between the variety of artefacts throughout the analysis process.

We believe these modifications serve to realise our aims of a fully unified methodology that considers both security and safety issues as first-class citizens, as well as providing significant traceability for ensuring the origin and development of critical requirements are clear.

### *3.2 Development from initial concept*

The authors have already put forward a prototypical version of the methodology (see Howard et al., 2017) in which the methodology had integrated the formal model into the analysis process at key points, but the security analysis aspect was not fully proven. We found that safety dominated the analysis process and that many of the critical requirements generated were ultimately safety-centric and that many security issues did not arise easily where they might have been entirely divorced from any safety impacts. As an example, exposure of information contained within a controller's process model may not pose any significant safety risk in isolation, but may represent an unacceptable breach of security because it may

further the understanding that an attacker is developing in preparation for an attack on the system.

The initial version of the methodology took the view that 'many security issues revealed by the analysis are likely to be a result of the causal analysis rather than the control action analysis' (ibid.) but it was clear from carrying out even rudimentary case studies that the causal factors analysis did not adequately address the innate difference between security and safety analysis, which is around the nature of the hazards.

Addressing this involved a review of security analysis methodologies within the literature and identification of common steps/ideas therein. This guided significantly the development of the security aspect of the analysis while attempting to maintain the principles that our methodology should be:

- Sufficiently high-level such that it could be applied at as much of the system life-cycle as possible.
- Particularly applicable to cyber-physical systems, particularly those in a national infrastructure or critical-systems context.
- Providing robust traceability from identification of a given security issue to critical requirement, as well as any associated formal method representations that demonstrate the completeness of the requirement.

Based on these principles, we developed our concept of *adversarial modelling* which sits between more formal representations of adversarial behaviours such as the Dolev-Yao model (Dolev and Yao, 1983) and more implementation-focused approaches such as misuse cases (Sindre and Opdahl, 2000) or CORAS (Braber et al., 2007). An important point to note is that our analysis does not seek to replace analyses such as formal security protocol analysis but instead seeks to allow for security (and safety) issues inherent within the requirements or the design of the system to be highlighted and corrected as early as possible (when it is far less expensive and much more robust to do so) than at a later point when one may have committed to specific protocols or implementations.

More detail on the actual process of adversarial modelling appears in subsubsection 4.2.6.

## 4 An overview of the methodology

### *4.1 Clarification on the illustrative example*

The illustrative example used to highlight each step of the methodology within subsection 4.2 is based on an abstracted and simplified version of the end-to-end technical architecture of the UK's smart meter system (Department of Energy & Climate Change, 2015). The system therefore consists of a multitude of smart meters on consumers' premises, as well as a central system which they report to.

### *4.2 Methodology steps*

#### *4.2.1 Step 1 - Establishing the system engineering basis*

We begin 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 our 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 what we would categorise as a system *Losses* which essentially represents 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 our 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 we seek to develop mitigations against. As per the STAMP model, one cannot control 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 our example is *"Meter registration is not correctly recorded by the system"*. It is easiest to represent these in a table - for extremely large collections of purposes, losses and hazards, it is easier 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 1.

| 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.<br>**H2:** Meter retirements are not actioned by the system. |

**Table 1**: Example purposes, hazards and losses mappings.

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. We seek, 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.

### *4.2.2 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. We do not seek to be too explicit in what exactly represents commands versus feedback, as this can be unduly restrictive.

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 our smart meter example, an exemplar control structure for it is given in Figure 1. 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.

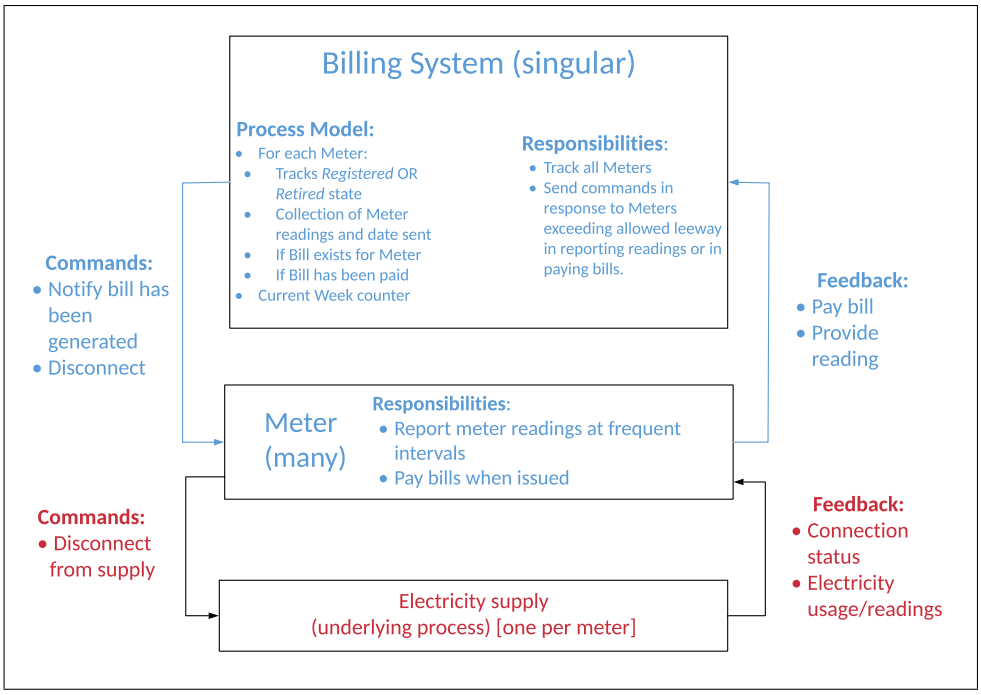


**Figure 1**: Functional control structure for smart meter example.

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 simply through annotations on each controller/process as has been used in our example.

### *4.2.3 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 us to create an initial formal model of the system in the next step.

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

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

Once all control actions have been identified, we can continue with the next phase of the analysis.

### *4.2.4 Step 4 - Building the initial formal model*

At this stage, we can construct an initial formal model of our 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. We seek to model sensible constraints on the system where these may already exist in requirements or practically 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.

We have not provided the formal model in the interest of conciseness but a simple model was created in Event-B which essentially represented all of the functionality displayed in the functional control structure (as shown in Figure 1).

### *4.2.5 Step 5 - Control action analysis and identification of critical requirements*

The control actions as determined in Step 3 (subsubsection 4.2.3) 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 our example, one can take the 'Register Meter' control action and subject it to analysis as is shown in Table 2 .

| **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. |

**Table 2**: Control action analysis results

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, we do not directly generate critical requirements for system-level hazards as these would essentially be vague and not beneficial to future stages of the system life-cycle. Instead, we seek to identify all ways that hazardous control can produce contributory hazards towards system-level hazards, and we seek to mitigate the hazardous control through 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 3.

| **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. |

**Table 3**: Critical requirement generation from our example.

We will integrate these critical requirements into the formal model in order to validate them in a later step.

### *4.2.6 Step 6 - Adversary modelling and generation of further critical requirements*

The next aspect of our analysis involves the notion of *adversarial 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.

The system's functional control structure itself must also be annotated with *manipulation points* - essentially these are any communication links between controllers, or controllers and processes, 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 indication that a given type of adversary has been considered, and has had sufficient mitigations against their behaviours in the form of critical requirements.

Going back to our example, we therefore have the annotated version of the functional control structure for the smart meter system in Figure 2, as well as a table containing the definition of an adversary in Table 4.

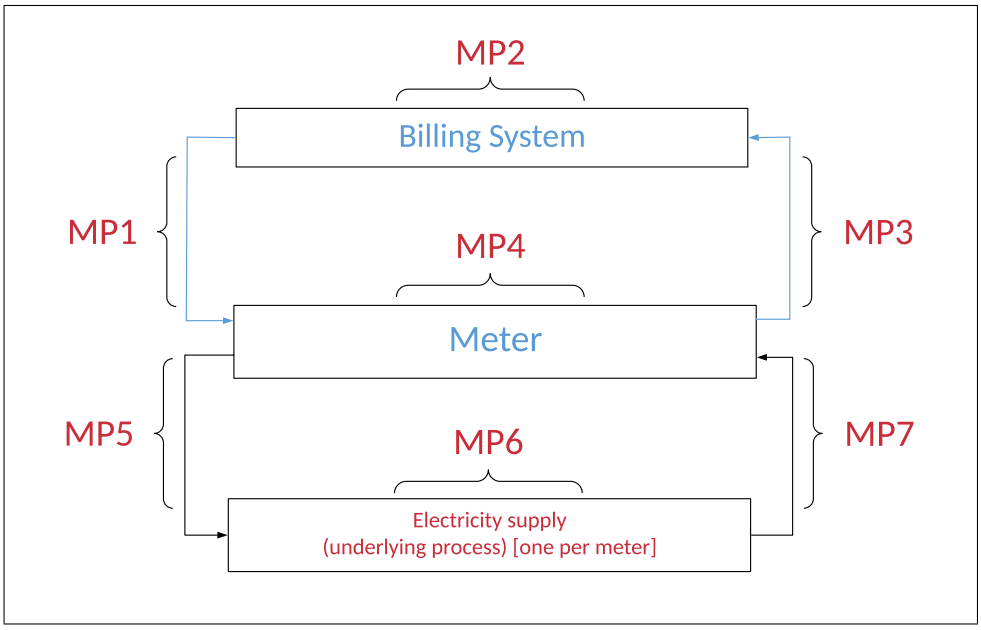


**Figure 2**: Annotated functional control structure for smart meter, with manipulation points.

| ***Detail*** | *Example adversary 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 meter out of sync with the underlying process. |

**Table 4**: Adversary description from the smart meter example.

We take the view that *adversary actions* are analogous to *hazards* as we cannot directly control the adversary or their actions. 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. We therefore seek 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 simply viewing it as a mere causal factor.

To generate critical requirements, we can use a summary of the adversary behaviours in lieu of the hazard column used in Table 3 and once again reuse the table to generate critical requirements. For hazards that have originated from the adversarial 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, we present some hazards and their associated critical requirements in Table 5.

**Table 5**: Critical requirement generation from our 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.<br>**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 we have collected a set of critical requirements through this generation process, we 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. We have in our example identified new commands between the two controllers in the form of the 'flagging' and the 'alert' control actions that are implicitly defined by our critical requirements and so this could inform a change to the functional control structure, which would necessitate beginning again.

### *4.2.7 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. We integrate each *critical requirement* 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:

*MetersRetireOverlapInvariant:*
$$RegisteredMeters \cap RetiredMeters = \varnothing$$

- Addition of guards to events to narrow the circumstances in which they may occur. We can restrict the RegisterMeter event through the guard below to ensure that retired meters may not be re-registered, for instance:

*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 our illustrative example, we determined that the use of 'tokens' were a sufficiently-capable abstraction of the notion of encryption between a meter and the billing system. We therefore have to store a valid token when registering a meter as demonstrated by the following action:

*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. Our 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:

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

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

VARIABLES:
$AllTokens$
$RegisteredTokens$

Additionally we made extensive use of event extension in our refinements 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.

### *4.2.8 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 *adversarial modelling* as security is often framed as being an issue of "intentional actions by malevolent actors" (Young and N. G. Leveson, 2014) 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" (ibid.) 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 of 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.

Going back once more to the example, we can take the hazard of "A retired meter is re-registered" as a hazard to subject to this analysis - this can be seen in Figure 3. We have annotated controllers 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. We then address the causal factors in Table 6.

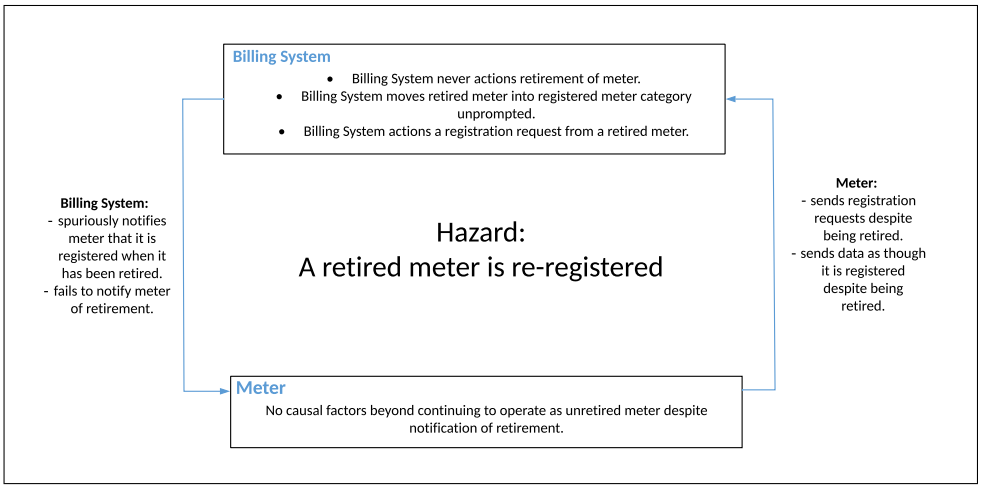


**Figure 3**: Causal factor analysis of one hazard from the example.

| **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. |

**Table 6**: Causal factors and the resulting critical requirements/design changes

This example is not the best example of the causal factors analysis process - if a human operator was included, acting 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.

### *4.2.9 Step 9 - Iteration and re-scoping*

One key element of the STPA family of methodologies is their focus on not simply carrying out analysis as a one-off effort but as a continuous process to iterate the design to a state where as much of the hazardous behaviour is mitigated against as possible within the system design. It may be that the identification of the hazardous behaviours and the resultant critical requirements indicate that the underlying system architecture requires a significant review. In this circumstance, it would be appropriate to begin this process from the beginning as any design changes will likely modify the functional control structure or the control actions within the system under analysis, which will necessitate beginning again. This analysis can

additionally be begun again from the initial step but be scoped in down to specific subsystems or portions of the functional control diagram to generate component-level hazards.

Analysing in this way allows for component-level losses, hazards and critical requirements to be identified but also allows the analyst to connect these artefacts of the analysis to those of the system as a whole. This is further supported by the formal model's support for decomposition of the overall abstract model into specific models for each component.

In the context of our example, it may therefore be useful to further scope down the analysis to individually analyse the meter and billing system individually, to consider candidate designs/architectures for how these entities may function that is less abstract and closer to an implementation-sufficient model. Each iteration of the system design should eventually result in a system with an adequate number of innate design elements that mitigate many identified hazards, as well as a set of critical requirements, for those hazards that cannot be trivially mitigated through design changes, to take forward to guide implementation work.

Our adversarial modelling also indicated the creation of new control actions in the critical requirements designed to mitigate against our example adversary, and so this is a valid reason to begin the analysis once more from the beginning with these design changes included. There are also suggested changes within the causal factors analysis that may inspire design changes.

## 5 A multi-UAV case study

To validate our methodology, in particular the developments described in subsection 3.2, it was applied to a further case study based on an existing paper which related to the coordination and control of multiple unmanned aerial vehicles (Bogdiukiewicz et al., 2017). The key findings from this case study are summarised below:

- The adversarial modelling concept allowed for more focus to be given to the security side of the analysis as the case study involved a multitude of entities communicating - reasoning about adversaries of different capabilities, access and intent was useful for identifying some changes to the system design in order to mitigate against many of the attack vectors.

- The formal model once more allowed for the generated critical requirements to be verified as they were applied to refinements of the model. Where critical requirements may have been inadequate to mitigate against their hazards completely, the use of model simulation and its generation of counter-examples aided in developing the critical requirements in such a way that they did become adequate in mitigating the identified hazards.

- We identified that there is some improvement to be undertaken on the use of adversarial modelling during the iteration/scoping of stage of the analysis as performing parallel analysis on sub-systems sometimes means that the adversary cannot be fully modelled. This is due to the fact that scoping the analysis purely to a sub-system can mean the full spectrum of adversarial actions cannot be modelled as the other components they may manipulate will be 'out-of-scope'. We were able to continue the analysis by either considering all inputs from other out-of-scope sub-systems to be compromised

by default or alternatively isolating the sub-system such that we did not consider any inputs or outputs as being in scope.

This could have been avoided by *not* scoping so narrowly down to sub-system level, but simply iterating the design to make it more concrete and continuing the analysis on the system as a whole. This suggests that the analysis may struggle to scale with larger, more complex systems where the sub-systems are themselves fairly complicated.

- More work needs to be undertaken in the area of formally modelling some common security properties of the system using Event-B. It should be possible to utilise the Event-B theory plug-in for Rodin (Butler and Maamria, 2013) to model concepts such as secrecy or for demonstrating reasoning about asymmetric encryption as two examples. There has been some existing work in the literature on the use of refinement in both B (Butler, 2002) and other formal methods such as Isabelle/HOL (Sprenger and Basin, 2010) which will be explored in order to improve this area of the analysis.

Overall, we found the case study to be useful in exploring the existing improvements made to the methodology, and additionally in contributing to potential future improvements to the methodology.

## 6 Conclusions and future work

In conclusion, we have presented a methodology which combines security and safety analysis in an integrated fashion in order to produce a set of *critical requirements* which are traceable to the hazards/adversary actions they seek to mitigate against and have their validity demonstrated through the formal model. The use of the formal model seeks to provides a higher degree of assurance that the critical requirements actually meaningfully mitigate against the hazards that they are intended to address. Our application of the methodology to both the smart meter example case study and the UAV case study indicate that the methodology allows for both safety and security to be considered within a single methodology in a meaningful way, and that the resultant critical requirements can be useful in guiding design iterations in order to reduce the chance of both hazardous control and adversarial action ultimately leading the system into a loss state.

Future work will focus on applying the methodology to further case studies in the critical infrastructure domain in order to guide refinement and improvements to the methodology and to bolster any areas in which the methodology may not be fully capable at present, such as the previously identified need for further work in modelling security properties, as well as producing supplementary tooling to support the analysis process from end-to-end.

## References

Abrial, Jean-Raymond (2010). *Modeling in Event-B: System and Software Engineering*. Cambridge University Press. ISBN: 9780521895569. DOI: DOI : 10 . 1017 / CBO9781139195881.

Behm, Patrick et al. (1999). "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, pp. 369–387. ISBN: 978-3-540-48119-5. DOI: 10.1007/3-540-48119-2_22.

Bogdiukiewicz, Chris et al. (Oct. 2017). "Formal Development of Policing Functions for Intelligent Systems". In: *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 194–204. DOI: 10.1109/ISSRE.2017.40.

Braber, F. den et al. (Jan. 2007). "Model-based security analysis in seven steps - A guided tour to the CORAS method". In: *BT Technology Journal* 25.1, pp. 101–117. ISSN: 13583948. DOI: 10.1007/s10550-007-0013-9.

Butler, Michael (2002). "On the use of data refinement in the development of secure communications systems". In: *Formal Aspects of Computing* 14.1, pp. 2–34. ISSN: 09345043. DOI: 10.1007/s001650200025.

Butler, Michael and Issam Maamria (2013). "Practical Theory Extension in Event-B". In: *Theories of Programming and Formal Methods: Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*. Ed. by Zhiming Liu, Jim Woodcock, and Huibiao Zhu. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 67–81. ISBN: 978-3-642-39698-4. DOI: 10.1007/978-3-642-39698-4_5.

Cai, H. et al. (2014). "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, pp. 2.1.2–2.1.2. ISBN: 978-1-84919-913-1. DOI: 10.1049/cp.2014.0993.

Colley, John and Michael Butler (Feb. 2013). "A Formal, Systematic Approach to STPA using Event-B Refinement and Proof". In: *21th Safety Critical System Symposium*.

Deng, Mina et al. (Mar. 2011). "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements". In: *Requirements Engineering* 16.1, pp. 3–32. DOI: 10.1007/s00766-010-0115-7.

Department of Energy & Climate Change (2015). *Smart Metering Implementation Programme - End to End Technical Architecture*.

Dhillon, B S (1992). "Failure modes and effects analysis — Bibliography". In: *Microelectronics Reliability* 32.5, pp. 719–731. ISSN: 0026-2714. DOI: https://doi.org/10.1016/0026-2714(92)90630-4.

Dolev, Danny and Andrew C. Yao (Mar. 1983). "On the Security of Public Key Protocols". In: *IEEE Transactions on Information Theory* 29.2, pp. 198–208. ISSN: 15579654. DOI: 10.1109/TIT.1983.1056650.

Dunjó, Jordi et al. (2010). "Hazard and operability (HAZOP) analysis. A literature review". In: *Journal of Hazardous Materials* 173, pp. 19–32. DOI: 10.1016/j.jhazmat.2009.08.076.

Friedberg, Ivo et al. (2017). "STPA-SafeSec: Safety and security analysis for cyber-physical systems". In: *Journal of Information Security and Applications* 34, pp. 183–196. ISSN: 22142126. DOI: 10.1016/j.jisa.2016.05.008.

Gawanmeh, Amjad et al. (Mar. 2012). "Formal Verification of Secrecy in Group Key Protocols Using Event-B". In: *Int. J. Communications, Network and System Sciences* 5.03, pp. 165–177. ISSN: 1913-3715. DOI: 10.4236/ijcns.2012.53021.

Howard, G. et al. (2017). "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*. ISBN: 9780769561073. DOI: 10.1109/EuroSPW.2017.68.

Kumar, Rajesh and Marielle Stoelinga (2017). "Quantitative security and safety analysis with attack-fault trees". In: *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, pp. 25–32. ISSN: 15302059. DOI: 10.1109/HASE.2017.12.

Künnemann, Robert and Graham Steel (2013). "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, pp. 257–272. ISBN: 978-3-642-38004-4. DOI: 10.1007/978-3-642-38004-4_17.

Lee, W. S. et al. (Aug. 1985). "Fault Tree Analysis, Methods, and Applications - A Review". In: *IEEE Transactions on Reliability* R-34.3, pp. 194–203. ISSN: 15581721. DOI: 10.1109/TR.1985.5222114.

Leveson, Nancy (Apr. 2004). "A new accident model for engineering safer systems". In: *Safety Science* 42.4, pp. 237–270. ISSN: 09257535. DOI: 10.1016/S0925-7535(03)00047-X.

Leveson, Nancy G. (2011). *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press, p. 555. ISBN: 9780262016629. DOI: 10.1017/CBO9781107415324.004.

Newcombe, Chris et al. (2015). "How Amazon web services uses formal methods". In: *Communications of the ACM* 58.4, pp. 66–73. ISSN: 00010782. DOI: 10.1145/2699417.

Procter, Sam, Eugene Y. Vasserman, and John Hatcliff (2017). "SAFE and Secure". In: *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*. New York, New York, USA: ACM Press, pp. 1–10. ISBN: 9781450352574. DOI: 10.1145/3098954.3105823.

Rasmussen, Jens (Nov. 1997). "Risk management in a dynamic society: a modelling problem". In: *Safety Science* 27.2-3, pp. 183–213. ISSN: 09257535. DOI: 10.1016/S0925-7535(97)00052-0.

Raspotnig, Christian et al. (Sept. 2013). "Enhancing CHASSIS: A method for combining safety and security". In: *Proceedings - 2013 International Conference on Availability, Reliability and Security, ARES 2013*. IEEE, pp. 766–773. ISBN: 9780769550084. DOI: 10.1109/ARES.2013.102.

Rezazadeh, Abdolbaghi et al. (2007). "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)*, pp. 1–8.

Schneier, Bruce (1999). "Attack Trees". In: *Dr. Dobb's Journal of Software Tools* 24.12, p. 60. ISSN: 1044-789X.

Sindre, G and A L Opdahl (2000). "Eliciting security requirements by misuse cases". In: *Proceedings 37th International Conference on Technology of Object-Oriented Languages and Systems. TOOLS-Pacific 2000*, pp. 120–131. ISBN: 1530-2067 VO -. DOI: 10.1109/TOOLS.2000.891363.

Sprenger, Christoph and David Basin (2010). "Developing security protocols by refinement". In: *Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, p. 361. ISSN: 15437221. DOI: 10.1145/1866307.1866349.

Thomas, John (2013). “Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis”. PhD thesis. Massachusetts Institute of Technology, p. 232. DOI: 10.2172/1044959.

Young, William and Nancy G. Leveson (Feb. 2014). “An integrated approach to safety and security based on systems theory”. In: *Communications of the ACM* 57.2, pp. 31–35. ISSN: 00010782. DOI: 10.1145/2556938.