Pattern-Based Formal Approach to Analyse Security and Safety of Control Systems

Vistbakka, Inna; Troubitsyna, Elena

Published in:
Model-Based Safety and Assessment. IMBSA 2019

DOI:
10.1007/978-3-030-32872-6_24

Publicerad: 01/01/2019

Document Version
(Referentgranskad version om publikationen är vetenskaplig)

Document License
Publisher rights policy

Pattern-Based Formal Approach to Analyse Security and Safety of Control Systems

Inna Vistbakka[1] and Elena Troubitsyna[1,2]

[1] Åbo Akademi University, Turku, Finland
[2] KTH – Royal Institute of Technology, Stockholm, Sweden
inna.vistbakka@abo.fi, elenatro@kth.se

Abstract. Increased openness and interconnectedness of safety-critical control systems calls for techniques enabling an integrated analysis of safety and security requirements. Often safety and security requirements have intricate interdependencies that should be uncovered and analysed in a structured and rigorous way. In this paper, we propose an approach that facilitates a systematic derivation and formalisation of safety and security requirements. We propose the specification and refinement patterns in Event-B that allow us to specify and verify system behaviour and properties in the presence of both accidental faults and security attacks and analyse interdependencies between safety and security requirements.

1 Introduction

Modern industrial control systems are rapidly becoming increasingly open and interconnected. Reliance on networking technologies offers a number of business benefits – flexibility, possibility to integrate new components and subsystems, remote control and diagnostics – just to name a few. However, the networked control systems are also becoming vulnerable to security threats. Security vulnerabilities can be exploited to undermine safety, e.g., by tampering with sensor data or hijacking the controlling functions.

Traditionally safety and security engineering have been considered to be two separate disciplines with different sets of methods and tools. Security analysis is typically data-centric, i.e., it focuses on determining the impact of security attacks on the system data flow. In contrast, safety analysis is concerned with defining the impact of failures on system functioning. Moreover, safety and security goals might result in the orthogonal functional requirements that are hard to resolve at the implementation level. Hence, there is a clear need for the modelling techniques that enable a formal reasoning about safety and security interdependencies at the early stages of the system development. In this work, we present a formal approach that allows the designers to uncover the implicit security requirements that are implied by the explicit system-level safety goals.

To analyse the intricate interdependencies between the requirements, we rely on formal modelling in Event-B [1]. Event-B is a rigorous approach to correct-by-construction system development by refinement. System development usually starts from an abstract specification that models the most essential system

functionality. In the refinement process, the abstract model is transformed into a detailed specification. While refining the system model, we can explicitly represent both nominal and failure behaviour of the system components as well as define the mechanisms for error detection and recovery. We can also explicitly represent the effect of security vulnerabilities such as tampering, spoofing and denial-of-service attacks and analyse their impact on system safety.

In our previous works [17, 18], we investigated the possibility to combine a traditional safety analysis approach and data flow analysis, while in [19] we studied application of Event-B and its refinement technique to uncover the interdependencies between safety and security. In the current paper, we extend and generalise our approach. We propose specification and refinement modelling patterns in Event-B to analyse security and safety requirements of control systems. These patterns capture the dynamic nature of safety and security interplay, i.e., they allow the designer to analyse the impact of deploying the security mechanisms on safety assurance and vice versa. An illustration of the proposed patterns is described in the formal development of a water treatment control system.

2 Safety and Security Interplay in Control Systems

In this section we discuss a generic architecture of a networked control system. We use the four-variable model of software-controlled systems proposed by Parnas [8]. This model (shown in Fig.1) defines the dependencies between the controlled physical process, input and output devices, and controller. The system goal is to maintain a physical process within the predefined safety bounds. The input device (sensor) measures the value of the controlled parameter that characterises the physical process. Then the controller reads this measurement as an input and computes the output – the state of the actuator. According to this state, the actuator affects the behaviour of the controlled physical process.

By applying the four-variable model, we derive two main types of requirements that should be implemented to guarantee system safety. The first type is the fault tolerance requirements. Since both sensors and actuators can be unreliable, to cope with their failures, either the system should contain redundancy or the controller should be able to put the system in a failsafe state. Moreover, the controller should consider the sensor imprecision. The second type of the requirements is correctness. We should guarantee that the controller output preserves the safety boundaries of the monitored physical process.

To address a connectivity of modern control systems, the four-variable model can be extended to take into account the impact of malicious attacks on the communication channels. Fig.2 presents our proposal to extend the four-variable model to define a generic architecture of a networked control system.

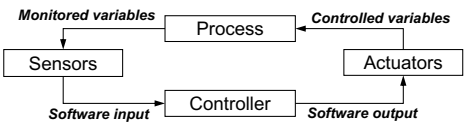

Fig. 1. The four-variable model

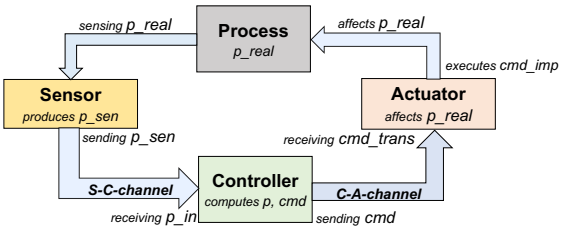

Fig. 2. Architecture of a control system and involved data control cycle

Let us discuss the behaviour of a networked control system and its components as well as a flow of the involved data. The controlled parameter, characterising the physical process, is denoted by *p_real*. The sensor senses the value of *p_real* and produces *p_sen*. Since, the sensor has a certain imprecision, i.e., the reading *p_sen* does not exactly match *p_real*. The measured value *p_sen* then is transmitted over the network to the the controller. In general, the transmission channel between the sensor and the controller *S-C-channel* might be untrusted, i.e., it might be a subject of security attack. Then the value that is received by the controller *p_in* might be different from *p_sen*.

The controller checks the reasonableness of the received *p_in*. It decides to use it as the current estimate of *p_real* or ignore it. The value p that the controller adopts as its current estimate of the process state should pass the feasibility check, i.e., should coincide with the predicted value and the freshness check, i.e., should be ignored if the transmission channel is blocked due to a DOS attack. If the controller ignores the received value *p_in*, it uses the last good value and the maximal variation of the process dynamics to compute p.

The value of p is then used to calculate the next state of the actuator that affects the controlled process. The command from the controller to the actuator is transmitted over a network. In the similar way, the transmission channel *C-A-channel* might be attacked. Hence, the command *cmd_trans* received by the actuator might be different from the command *cmd* send by the controller. Upon receiving the command *cmd_trans* the actuator applies it, which should result in the desirable change of the process state.

For our generic control system, we can define the main safety requirement as the following predicate: $Safety = p\_real \leq safe\_threshold \vee stop{=}TRUE$. It means that the controlled process shall be kept within the safety bounds while the system is operational; otherwise, a safe shutdown should be executed. While designing a networked control system, our goal is to proof its *Safety*.

Traditionally, the design of a safety-critical software-intensive control system relies on specific assumptions and properties of the domain as well as properties of the controlling software [19]. During the design of such a system we should prove the following judgement:

$$(\ \boldsymbol{ASM,\ DOM,\ SW}\ ) \vdash Safety,$$

Here ***ASM, DOM*** and ***SW*** stand for assumptions, domain and controlling software properties, respectively. Next we will discuss these three types of properties that suffice to proof *Safety* for our generic control system.

– ***ASM*** – *assumptions:*
A1. $p\_sen = p\_real \pm \Delta_1$
A2. $p = p\_sen \ + \ \Delta_2 \wedge \Delta_2 = k\Delta_3$
A3. $(stop = FALSE \ \wedge \ cmd\_trans = cmd) \vee \ stop = TRUE$

The assumption ***A1*** means that the sensor measurements are sufficiently precise and unprecision is bounded, where Δ_1 is its maximal imprecision value. The assumption ***A2*** states that the controller always adopts a measurement of the value of the process parameter that either coincides with *p_sen*, i.e., $k = 0$, or is calculated on the basis of the last good value and Δ_3 – the maximal possible increase of the value *p_real* per cycle ($\Delta_2 = k\Delta_3$, where k is the number of cycles). This assumption implies both safety and security requirements. Firstly, we should guarantee that the channel *S-C-channel* is tamper resistant and the sensor is spoofing resistant. Secondly, we should ensure that the controlling software checks the validity of the input parameter and ignores it, if the check fails. The assumption ***A2*** also implies that, in case of an attack on the channel *S-C-channel*, the system continues to function for some time by relying of the last good value. The assumption ***A3*** means that if a failure or an attack on the channel *C-A-channel* is detected then the system is shut down. It means that the system should have some (possibly non-programmable) way to execute a shutdown in case the channel *C-A-channel* becomes unreliable.

– ***DOM*** – *domain properties:*
D1. $cmd = INCR \Rightarrow p\_real_{c+1} \geq p\_real_c$ (for any system cycles c and $c + 1$), while system is operational.
D2. $cmd = DECR \Rightarrow p\_real_{c+1} < p\_real_c$, while system is operational
D3. $max|(p\_real_{c+1} - p\_real_c)| = \Delta_3$
D4. $failsafe{=}TRUE \Rightarrow p\_real_{c+1} \leq p\_real_c$, while system is shut down.

The property ***D1*** states that an execution of the command *incr* results in the increase of the value *p_real*. The property ***D2*** is similar to ***D1***. The property ***D3*** states that the maximal possible increase of *p_real* per cycle is known and bounded. ***D4*** stipulates that when the system is put in the failsafe state, the value of the physical parameter does not increase.

– ***SW*** – *controlling software property*
S1. $p\_est + \Sigma_{i=1}^{3}\Delta_i \geq safe\_threshold \wedge stop{=}FALSE \Rightarrow cmd = DECR$

Here, the software property ***S1*** corresponds to the safety invariant that controller should maintain: the controller issues the command *decr* to the actuator if at the next cycle the safe threshold can be exceeded.

Straightforward logical calculations allow us to prove

$$(\textbf{A1, A2 , A3, D1, D2, D3, D4, S1}) \vdash Safety.$$

Our system level analysis has demonstrated that both safety and security aspects are critical for fulfilling the system-level goal of ensuring safety. Hence, both these aspects should be explicitly addressed during the system development. It is easy to observe, that we had to define a large number of requirements even for a generic high-level system architecture. To facilitate a systematic requirements derivation, we propose to employ formal development framework Event-B.

3 Modelling and Refinement in Event-B

Event-B [1] is a state-based framework that promotes the correct-by-construction approach to system development and formal verification by theorem proving. In Event-B, a system model is specified using the notion of an *abstract state machine*. It encapsulates the model state, represented as a collection of variables, and defines operations on the state, i.e., it describes the dynamic behaviour of a modelled system. A machine has an accompanying component, called *context*, which includes user-defined sets, constants and their properties given as axioms.

The dynamic behaviour of the system is defined by a set of atomic *events*. Generally, an event has the following form:

$$e \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } G_e \textbf{ then } R_e \textbf{ end},$$

where e is the event's name, a is the list of local variables, G_e is the event guard, and R_e is the event action. The guard is a predicate over the local variables of the event and the state variables of the system. The guard defines the conditions under which the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

In general, the action of an event is a parallel composition of deterministic or non-deterministic assignments. A deterministic assignment, $x := E(x, y)$, has the standard syntax and meaning. A non-deterministic assignment is denoted either as $x :\in S$, where S is a set of values, or $x : |P(x, y, x)$, where P is a predicate relating initial values of x, y to some final value of x. As a result of such an assignment, x can get any value belonging to S or according to P.

Event-B employs a top-down refinement-based approach to system development. Development typically starts from an abstract specification that nondeterministically models the most essential functional requirements. In a sequence of refinement steps, we gradually reduce nondeterminism and introduce detailed design decisions. In particular, we can add new events, split events as well as replace abstract variables by their concrete counterparts, i.e., perform *data refinement*. When data refinement is performed, we should define *gluing invariants* as a part of the invariants of the refined machine. They define the relationship between the abstract and concrete variables.

The consistency of Event-B models, i.e., verification of well-formedness, invariant preservation and correctness of refinement steps, is demonstrated by discharging a number of verification conditions – proof obligations. The Rodin platform [2] provides an automated support for formal modelling and verification in Event-B. It automatically generates the required proof obligations and attempts to discharge (prove) them automatically.

4 Pattern-Based Development of a Control System in Event-B

In this section, we present a generic methodology for the refinement-based development of control systems that facilitates identifying implicit security requirements that should be fulfilled to satisfy system safety. To support such development of a control system in Event-B, we define a set of Event-B specification

and refinement patterns that reflect the main concepts of the safety-security co-engineering discussed in the previous section. Such patterns represent generic modelling solutions that can be reused in similar developments.

4.1 Specification and Refinement Patterns

Control Cycle Modelling Pattern. This pattern corresponds to the initial Event-B specification. To formulate this pattern, we introduce an abstract type $PHASES$={$PROC$, SEN, TO_CONTR, $CONTR$, TO_ACTUA, $ACTUA$} defining all stages of a control cycle. Moreover, a variable $phase \in PHASES$ abstractly models a current stage of a control cycle.

The abstract model (given in Fig.3) represents the overall behaviour of the control system by a set of events modelling the phases of a control cycle. In the initial model, we also abstractly specify an occurrence of faults. The event **FailureDetection** non-deterministically models the outcome of the error detection. A reaction on errors is abstractly modelled by the event **FailSafe**.

Physical Process Modelling Pattern. The objective of this modelling pattern is to explicitly introduce the behaviour of the environment – introduce dynamics of the controlled process. While defining this pattern, we should ensure that the domain properties ***DOM***, discussed in the Section 2, are formalised.

We define the behaviour of the physical process characterised by the variable *p_real*. We also model the dependencies between the actuator state and the expected range of *p_real* value. The abstract function *process_fnc* (formulated in the model context) is used to specify our knowledge about the process:

$$process\_fnc \in \mathbb{N} \times ACTUATOR\_STATES \rightarrow \mathbb{N}.$$

This function takes as an input the previous value of the process as well as the actuator state and returns a next predicted value of the process. We also

Process $\widehat{=}$
when $phase{=}PROC \wedge stop{=}FALSE$
then $phase{:=}SEN$
end

Sensor $\widehat{=}$
when $phase{=}SEN \wedge failure{=}FALSE \wedge stop{=}FALSE$
then $phase{:=}TO\_CONTR$
end

S_C_Chan $\widehat{=}$ ...
when $phase{=}TO\_CONTR \wedge failure{=}FALSE$
then $phase{:=}CONTR$
end

Controller $\widehat{=}$
when $phase{=}CONTR \wedge failure{=}FALSE \wedge stop{=}FALSE$
then $phase{:=}TO\_ACTUA$
end

C_A_Chan $\widehat{=}$
when $phase{=}TO\_ACTUA \wedge failure{=}FALSE \wedge stop{=}FALSE$
then $phase{:=}ACTUA$
end

Actuator $\widehat{=}$
when $phase{=}ACTUA \wedge failure{=}FALSE \wedge stop{=}FALSE$
then $phase{:=}PROC$
end

FailureDetection $\widehat{=}$ ...

FailSafe $\widehat{=}$
when $phase{=}CONTR \wedge failure{=}TRUE \wedge stop{=}FALSE$
then $stop{:=}TRUE$
end

Fig. 3. Events of the *Control Cycle Modelling Pattern*

formulate the properties of the process dynamic depending on the actuator state:

$$\forall n \cdot n \in \mathbb{N} \Rightarrow process\_fnc(n \mapsto ON) \geq n,\ \forall n \cdot n \in \mathbb{N} \Rightarrow process\_fnc(n \mapsto OFF) \leq n.$$

When the actuator state is ON, the value of *p_real* should increase. Correspondingly, while the actuator state is OFF, the value of *p_real* should decrease. We formulate these properties in the model context. Thereby we formalize the properties ***D1*** and ***D2*** discussed in Section 2. Moreover, the following constraint in the context formalises the property ***D3***:

$$\forall\ n\ \cdot\ n \in 0\ ..\ p\_max + delta3 \Rightarrow process\_fnc(n \mapsto ON) \leq safe\_threshold$$

It requires that, if the process state is currently in the safe range [*0..p_max* + *delta3*], it cannot exceed the critical range within the next cycle, i.e., the safety gap between *p_max* and *safe_threshold* is sufficiently large.

We then refine the abstract event **Process** and model the changes of the physical process:

Process refines **Process** $\widehat{=}$
when *phase=PROC* $\wedge$ *stop=FALSE*
then *phase:=SEN*
p_real:=`process_fnc`*(p_real* $\mapsto$ *act_state)*
end

Sensor Behaviour Modelling Pattern. In this pattern we specify normal and faulty sensor behaviour as well as a detection of a sensor failure. In the refined machine, we introduce a variable *p_sen* to model the value of the physical variable measured by the sensor. It can be affected by the sensor imprecision or failures, thus our goal is also to specify the assumption ***A1***.

The event **Sensor_Normal** models the behaviour of the sensor by assigning to the variable *p_sen* any value from the range [*p_real - delta1 ... p_real + delta1*]. Here *delta1* is the maximal imprecision value for the sensor introduced as a model constant. The event **Sensor_Failure** models the sensor failure. In case of a failure, the sensor produces the reading that is out of the expected range.

Sensor_Normal refines **Sensor** $\widehat{=}$
when *phase=SEN* $\wedge$ *failure=FALSE* $\wedge$ *stop=FALSE*
then *phase:=TO_CONTR*
p_sen $:\in$ *p_real* $-$ *delta1 ... p_real* $+$ *delta1*
end
Sensor_Failure refines **Sensor** $\widehat{=}$
when *phase=SEN* $\wedge$ *failure=FALSE* $\wedge$ *stop=FALSE*
then *phase:=TO_CONTR*
p_sen $:|$ *p_sen*$'$ $\in \mathbb{N} \wedge$ *p_sen*$'$ $\notin$ *p_real* $-$ *delta1 ... p_real* $+$ *delta1*
end

Sensor's Data Transmission Modelling Pattern. This pattern aims at modelling sensor reading communication to the controller. We introduce the variable *p_in* denoting the value of the sensor measurement received by the controller as an input. It might differ from the *p_sen* value due to possible security attack on the channel *S_C_Channel*. We then refine the abstract event **S_C_Chan** by two more concrete events, modelling the normal and abnormal cases of data transmission, where we assign to *p_in* different outcomes.

S_C_Chan_Normal refines **S_C_Chan** $\widehat{=}$
when $phase{=}TO\_CONTR \wedge failure{=}FALSE \wedge attack\_s\_c{=}FALSE$
then $phase{:=}CONTR$
$p\_in := p\_sen$
end

S_C_Chan_Failure refines **S_C_Chan** $\widehat{=}$
when $phase{=}TO\_CONTR \wedge failure{=}FALSE \wedge attack\_s\_c{=}TRUE$
then $phase{:=}CONTR$
$p\_in :| (p\_in' \in \mathbb{N} \wedge p\_in' \neq p\_sen \wedge p\_in' \notin p\_real - delta1 \ldots p\_real + delta1)$
end

Moreover, to abstractly model a possible attack on the channel *S-C-channel*, we define a variable $attack\_s\_c \in BOOL$ indicating whether the system is under attack. The attack can happen anytime while transmitting the sensed data to the controller and is modelled by the event **Attack_S_C_Chan**. We use this abstraction to represent the results of the security monitoring.

At this refinement step, we prove the property that describes the effect of the attack on *S_C_Channel*: $attack\_s\_c{=}FALSE \wedge phase{=}CONTR \Rightarrow p\_in{=}p\_sen$.

Controller Behaviour Modelling Pattern. The goal of this pattern is to uncover a detailed specification of the controller behaviour (and also specify ***SW*** assumption). In this refinement step, we refine the abstract event **Controller** to represent different alternatives that depend on the received sensor reading.

We model the procedure of computing the current estimate p. The controller either accepts the current input or relies on the last good value, or calculates a new value p on the basis of the last good value and the maximal possible increase per cycle. The computed value of p is used to calculate the output – the next state of the actuator, i.e., update the variable *cmd*.

The output of the controller – the next state of the actuator – depends on the value of p adopted by the controller as the current estimate of the process state. Upon receiving the input *p_in* the controller checks its reasonableness. If the check is successful then p obtains the value of *p_in*. Then the controller proceeds by checks whether p exceeds *p_max* or is in the safe range *[0..p_max]*. These alternatives are modelled by the events **Controller_normal_DECR** and **Controller_normal**, correspondingly.

If the input does not pass the reasonableness check, the controller calculates the value of the process parameter using the last good input value and the maximal possible increase of the value *p_real* per cycle *delta3*. Then, the controller checks whether p exceeds *p_max* and computes the output. These alternatives are covered by the events **Controller_retry_DECR** and **Controller_retry**, correspondingly. Here the variable *retry* is introduced to model the number of retries before the failure is considered to be a permanent and system is shut down. The behaviour of the controller preserves the following invariants:

$$phase = TO\_ACTUA \wedge p > p\_max \Rightarrow cmd{=}DECR,$$

$$phase = TO\_ACTUA \wedge p \in 0..p\_max \Rightarrow cmd{=}INCR \vee cmd{=}DECR.$$

They postulate that the controller issues the command *DECR* if the parameter p is approaching the critically high value. If the controlled parameter is within the safety region then the controller output might be either *DECR* or *INCR*.

Controller_normal_DECR refines **Controller** $\widehat{=}$
when $phase{=}CONTR \wedge failure{=}FALSE \wedge stop{=}FALSE \wedge$
$p\_in = process\_fnc(p \mapsto act\_state) \wedge p\_in > p\_max$
then $phase{:=}TO\_ACTUA$
$p := p\_in$
$cmd := DECR$
$retry := 0$
end

Controller_retry_DECR refines **Controller** $\widehat{=}$
any $p\_new, delta3$
where $phase{=}CONTR \wedge failure{=}FALSE \wedge stop{=}FALSE \wedge$
$p\_in \neq process\_fnc(p \mapsto act\_state) \wedge retry \leq 2 \wedge delta2{=}(retry{+}1){*}delta3 \wedge$
$p\_new \in p{-}delta3 \ldots p{+}delta3 \wedge p\_in > p\_max$
then $phase{:=}TO\_ACTUA$
$p := p\_new$
$cmd := DECR$
$retry := retry + 1$
end

Controller's Command Transmission Modelling Pattern. The goal of this modelling pattern is to introduce into the Event-B model a transmission of the command issued by the controller to the actuator as well as introduce an abstract representation of the attacks and the system reaction on them.

We construct this pattern similarly to *Sensor's Data Transmission Modelling Pattern*. We add several new variables and events into the refined system specification, (e.g., *attack_c_a*, the events **Attack_C_A_Chan**, **C_A_Chan_Normal**, **C_A_Chan_Failure**). According to the assumption ***A3***, if an attack on the channel *C-A-channel* has occurred then the controller output would differ from the command received by the actuator. Safety cannot be ensured if an attack on the channel *C-A-channel* is detected and hence the system should be shut down.

We formulate and prove the following property, that describes the effect of the attacks on the controller output: $attack\_c\_a{=}FALSE \wedge phase{=}ACTUA \Rightarrow cmd{=}cmd\_trans$.

Actuator Behaviour Modelling Pattern. Our last refinement pattern focuses on modelling the behaviour of the actuator. We assume that the actuator can fail during performing its function. Then the impact produced by the actuator on the process might also deviate from the one associated with the command *cmd*. We define by the variable *cmd_imp* the state of the actuator produced on the process and assign to it different outcomes depending on the actuator behavior. We refine our abstract event **Actuator** and model different alternatives.

As a result of this refinement step, we arrive at a sufficiently detailed specification to define and prove the following safety invariant: $p\_real \in 0 \ldots safe\_threshold$.

Actuator_Normal refines **Actuator** $\widehat{=}$
when $phase{=}ACTUA \wedge failure{=}FALSE \wedge stop{=}FALSE \wedge attack\_c\_a{=}FALSE$
then $phase{:=}PROC$
$cmd\_imp := cmd$
end

Actuator_Failure refines **Actuator** $\widehat{=}$
when $phase{=}ACTUA \wedge failure{=}FALSE \wedge stop{=}FALSE \wedge attack\_c\_a{=}TRUE$
then $phase{:=}PROC$
$cmd\_imp :\in CMD \setminus \{cmd\}$
end

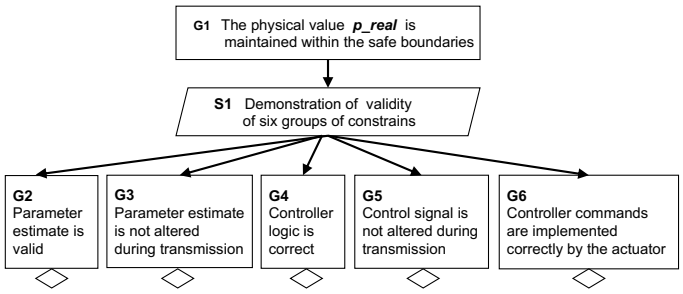

Fig. 4. Decomposition of top-level safety goal

4.2 Construction of Evidences for Safety Case

Safety-critical systems should be developed in such a way that their safety is also demonstrable, i.e., it can be convincingly argued that the system is acceptable safe. The safety argument – a *safety case* explicitly defines the safety requirements and justifies why the design adequately implements them. Goal Structuring Notation [5] has became a popular form of representing a safety case. It explicitly describes how the safety goals are decomposed into subgoals until claims can be supported by the direct evidences.

During our formal modelling we derive and verify safety and safety-related security requirements, then the artifacts collected during the development can be used in the safety case construction. Next we briefly demonstrate how to construct the evidence justifying the safety goal associated with a control system using different specifications and proofs constructed during the system development. The detailed guidelines for constructing the safety cases from the formal specification in Event-B are described in our previous work [12].

Fig. 4 depicts a part of the resulting safety case for our generic networked control system. Rectangles contain definitions of goals, parallelograms show the definitions of the strategies, while circles represent solutions. Lets consider the goal **G4** (Fig.5): "The controller logic is correct". It is considered in the context of formal modelling in Event-B with Rodin platform tool (**C1**). To support that claim **G4** holds, we state a strategy **S4** to be used in solution of a goal. Namely, we need to define constrains over constants as axioms. Moreover, we have to model the controller actions as well as define the safety invariant and prove it preservation during system execution. Consequently, we further decompose the goal **G4** into three subgoals and define the solutions that support the claims.

Next we will demonstrate how the proposed pattern-based refinement process can be applied to development of a water treatment control system [13].

5 Case Study: Water Treatment Control System

In this section, we overview a water control system [13] and briefly discuss how to develop an Event-B specification of this system and uncover the mutual interdependencies between safety and security requirements. While our modelling, we will rely on the generic development patterns presented in Section 4.

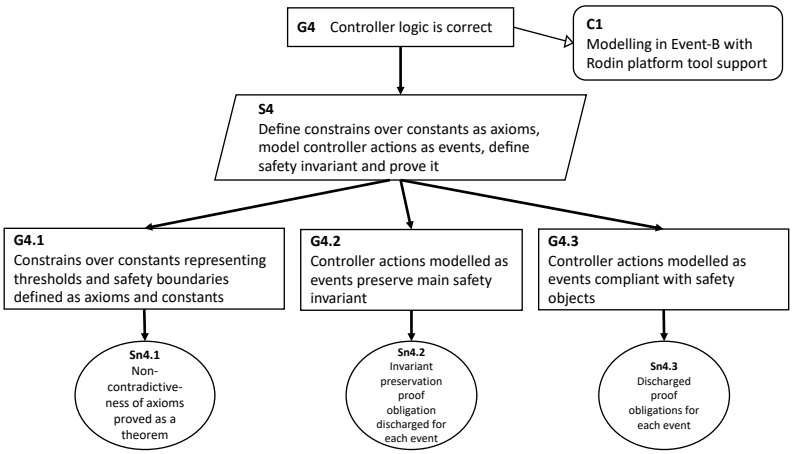

Fig. 5. Decomposition of G4 safety goal

5.1 Case Study Description

We consider a minimal set up in Modern Industrial Control System – a water treatment system [13]. The water treatment system (WTS) is a control system that adjusts the quantity of water in the tank to maintain it within the predefined safety bounds. The system consists of the following main components (depicted in Fig.6): motorized inflow valve, a tank, a pump, a sensor to measure the quantity of water in the tank, Programmable Logic Controller (PLC) and a central supervisory control system (SCADA).

The system performs the following global scenario. An inflow valve let passage of water into a tank through a pipe. A tank is equipped with a sensor which measures the level of the water inside the tank. Then the sensor communicates its reading to PLC. When the level of the water reaches a certain upper (lower) threshold, PLC communicates to the motorized inflow valve to close (open) and to the pump to start (stop). The sensor and actuators (pump, valve) operate by receiving and sending analog signals. PLC converts the analog signals into digital signals. The digital signals are then exchanged between PLCs and SCADA.

The main hazard of the system is associated with overflow of water in the tank. The main safety goal of WTS is then to keep the level of the water *WL_real* inside the tank within the predefined boundaries: $0 \leq WL\_real \leq WL\_max\_crit$.

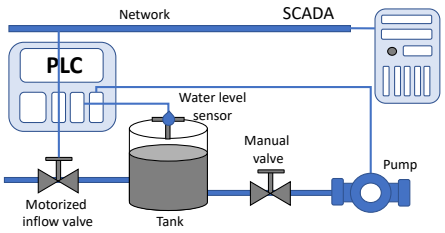

Fig. 6. Water treatment system

Since WTS is an example of the networked system then it could threaten to increase vulnerability to malicious attacks. In general case, we assume that the attackers goal is to cause a water burst in the tank. Therefore, while reasoning about the behaviour of such a system, we should also reason about the impact of security threats on its safety. The analysis presented in Section 2 shows that safety cannot be guaranteed when the controller-actuator channel is attacked. Then WTS should include an additional component – a manual valve – that should be placed in the systems architecture. The behavior of the manual valve is the same as the inflow valve. The only difference is that the manual valve can only be manually operated, i.e., cannot be operated using network messages. Such a non-programmable component can put the system in the failsafe state to guarantee safety. Next we present an abstract Event-B specification of WTS.

5.2 Event-B Development of the Water Treatment System

We design the Event-B specification of WTS incrementally, i.e., by gradually unfolding system functionality and architecture. Our development relies on the generic development presented in Section 4. Due to space limits, we only highlight the most important modelling solutions of the development.

Initial Specification. We start with an initial model of WTS where we define its cyclic behaviour. By following the guidelines defined in Section 4, we introduce an abstract representation of the control cycle and the corresponding phases.

The event WaterTank models the changes of the water level parameter *wl_real* while charging. WLSensor event models the estimation of this parameter (that is defined by *wl* variable). The event PLC specifies the PLC actions (i.e., sending the command to open or close valve and to stop pump). The events TransmReadings and TransmCommands model transmission of the corresponding sensor readings and controller commands, correspondingly. Finally, Actions event models the required actions upon receiving the commands from PLC.

First Refinement. At this step we elaborate on the dynamics of the controlled process, i.e., define the changes in the water level *wl_real*.

Second refinement. Here we model the behaviour of the sensor and unfold the value of the physical variable *wl_sen* measured by the sensor. Since this value can be affected by the sensor imprecision or failures, we address it in our model.

Third refinement. We model the transmission of the sensor reading to PLC.

Fourth refinement. This step aims at introducing a detailed specification of the PLC logic. We define the control algorithm, i.e., model the behavior of the controller. The controller calculates the commands to be send to the valve and pump using the current estimate of the water level.

At each control cycle, the PLC controller receives the current estimate of the water level from the sensor. The controller checks whether the water level is still in safe range and sends commands continue to open valve or close valve and stop the pump. The decision to continue water supply can be made only if the controller verifies that the water level at the end of the next cycle will still be in the safe range $[0 \ldots wl\_max\_crit]$.

We refine the abstract events of the previous refinement step to represent different alternatives of PCL behaviour. At this stage we can formulate correctness

of the WTS logic by the following invariants:

$$phase = TRANSM \wedge wl \geq wl\_max \Rightarrow signal{=}STOP,$$
$$phase = TRANSM \wedge wl < wl\_max \Rightarrow signal{=}CONT.$$

The invariants postulate that the WTS issues the signal to stop when the parameter *wl* is approaching the critically high value (*wl_max_crit*), and vice versa. To give the system a time to react, WTS sends the stopping command to the actuators whenever the value *wl* breaches the predefined value *wl_max*.

Fifth refinement. We model signal transmission issued by PCL to the valve and the pump. Here we model different cases of the nominal and abnormal signal transmission (including DOS attack, security failure, etc). We incorporate into the model architecture a certain mechanism that would allow the system to transmit the signal in a secure way. In particular, we add a new component – security gateway – between the WTS and the external actuators. It could control the network access according to predefined security policies and can also inspect the packet content to detect intruder attacks and anomalies.

Sixth refinement. Finally, we elaborate on the behaviour of the actuators. Upon receiving the command from PCL, the valve closes and the pump starts or valve opens. As a result of the last refinement step, we arrive at a sufficiently detailed formal specification to define and prove the desired system level property: $wl\_real \in 0 \mathinner{..} wl\_max\_crit$.

6 Related Work and Conclusions

In recent years, co-engineering of security and safety requirements has received increasing attention by researchers [10, 20, 16]. The possible integration of safety and security techniques has been addressed by adaptation conventional techniques for analysing safety risks to perform a security-informed safety analysis [4, 14]. Proposed techniques provide the engineers with a structured way to discover and analyse security vulnerabilities that have implications on system safety.

Formal analysis of safety and security requirements interactions has been addressed in works [6, 11]. However, most of the works focus on finding and demonstrating conflicts between safety and security. In contrast, in our work, we treat the problem of safety-security interplay at a more detailed level. We analyse the system architecture, investigate the impact of security failures on system functioning and system safety. Such an approach allows us to study the dynamic nature of safety-security interactions. Ponsard et al. [11] study how Goal-Oriented Requirements Engineering can support co-engineering to address the safety and security dimensions in cyber-physical systems.

In the work [3] distributed MILS approach is presented to support a powerful analysis of the properties of the data flow using model checking and facilitates derivation of security contracts. Since our approach enables incremental construction of complex distributed architectures, it would be interesting to combine these techniques to support an integrated safety-security analysis throughout the entire formal model-based system development.

The four-variable model proposed by Parnas has also been adopted in work [9]. The authors show application of this model in the development of safety-critical systems in industry. They show how this model helps to define the behaviours of, and the boundaries between, the environment, sensors, actuators, and controlled software. Similarly, in our work four-variable model allows us to derive the behaviour of controller that is acceptable from the safety point of view. However, we also employ formal modelling technique to uncover mutual interdependencies between safety and security.

Similar to our work, a model-based approach for the formalization of system requirements, and their early validation with respect to system design has been proposed in [15]. This work introduces a bottom-up design approach as opposed to the top-down approach that used in Event-B. The emphasis is on property preservation through gradual composition of models derived from patterns (architectures) rather than on refinement and generation of model invariants and constraints such as in the Event-B.

The approach to integrate the modelling with UML, formal methods and the actual implementation for developing security-critical applications is discussed in [7]. However, the authors do not consider safety aspect of the modelled system.

In this work, we have proposed a formal approach enabling derivation and formalising safety and security requirements using correct-by-construction development paradigm. Our proposed approach allows us in a systematic manner to derive the constraints that should be imposed on the system to guarantee its safety even in presence of the security attacks. Our approach has relied on modelling and refinement in Event-B.

The approach presented in this work generalises the results of our experience with formal refinement-based development in the Event-B conducted in the context of verification of safety-critical control system [17–19]. The results have demonstrated that the formal development significantly facilitates derivation of safety and security requirements. We have also observed that the integrated safety-security modelling in Event-B could be facilitated by the use of external tools supporting constraint solving and continuous behaviour simulation. Such an integration would be interesting to investigate in our future work.

References

1. Abrial, J.R.: Modeling in Event-B. Cambridge University Press (2010)
2. Abrial, J., Butler, M.J., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. STTT 12(6), 447–466 (2010), https://doi.org/10.1007/s10009-010-0145-y
3. Cimatti, A., DeLong, R., Marcantonio, D., Tonetta, S.: Combining MILS with Contract-Based Design for Safety and Security Requirements. In: SAFECOMP 2015 Workshops. LNCS, vol. 9338, pp. 264–276. Springer (2015), https://doi.org/10.1007/978-3-319-24249-1_23
4. Fovino, I.N., Masera, M., Cian, A.D.: Integrating cyber attacks within fault trees. Rel. Eng. & Sys. Safety 94(9), 1394–1402 (2009), https://doi.org/10.1016/j.ress.2009.02.020

5. Kelly, T.P., Weaver, R.A.: The goal structuring notation – A safety argument notation. In: DSN 2004, Workshop on Assurance Cases (2004)
6. Kriaa, S., Bouissou, M., Colin, F., Halgand, Y., Pièvre-Cambacédès, L.: Safety and security interactions modeling using the BDMP formalism: sase study of a pipeliLeve. In: SAFECOMP 2014. LNCS, vol. 8666, pp. 326–341. Springer (2014), https://doi.org/10.1007/978-3-319-10506-2_22
7. Moebius, N., Haneberg, D., Reif, W., Schellhorn, G.: A modeling framework for the development of provably secure e-commerce applications. In: (ICSEA 2007). p. 8. IEEE Computer Society (2007), https://doi.org/10.1109/ICSEA.2007.7
8. Parnas, D.L., Madey, J.: Functional documents for computer systems. Sci. Comput. Program. 25(1), 41–61 (1995), https://doi.org/10.1016/0167-6423(95)96871-J
9. Patcas, L.M., Lawford, M., Maibaum, T.: Implementability of requirements in the four-variable model. Sci. Comput. Program. 111, 339–362 (2015)
10. Paul, S., Rioux, L.: Over 20 years of research into cybersecurity and safety engineering: a short bibliography. Safety and Security Engineering VI(335) (2015)
11. Ponsard, C., Dallons, G., Massone, P.: Goal-oriented co-engineering of security and safety requirements in Cyber-Physical Systems. In: SAFECOMP 2016 Workshops DECS. pp. 334–345. Springer International Publishing (2016), https://doi.org/10.1007/978-3-319-45480-1_27
12. Prokhorova, Y., Laibinis, L., Troubitsyna, E.: Facilitating construction of safety cases from formal models in event-b. Information & Software Technology 60, 51–76 (2015), https://doi.org/10.1016/j.infsof.2015.01.001
13. Rocchetto, M., Tippenhauer, N.O.: CPDY: extending the Dolev-Yao attacker with physical-layer interactions. In: ICFEM 2016. LNCS, vol. 10009, pp. 175–192 (2016), https://doi.org/10.1007/978-3-319-47846-3_12
14. Schmittner, C., Ma, Z., Smith, P.: FMVEA for safety and security analysis of intelligent and cooperative vehicles. In: SAFECOMP Workshops 2014. LNCS, vol. 8696, pp. 282–288. Springer (2014), https://doi.org/10.1007/978-3-319-10557-4_31
15. Stachtiari, E., Mavridou, A., Katsaros, P., Bliudze, S., Sifakis, J.: Early validation of system requirements and design through correctness-by-construction. Journal of Systems and Software 145, 52–78 (2018), https://doi.org/10.1016/j.jss.2018.07.053
16. Troubitsyna, E., Laibinis, L., Pereverzeva, I., Kuismin, T., Ilic, D., Latvala, T.: Towards security-explicit formal modelling of safety-critical systems. In: SAFECOMP 2016. LNCS, vol. 9922, pp. 213–225. Springer (2016), https://doi.org/10.1007/978-3-319-45477-1_17
17. Troubitsyna, E., Vistbakka, I.: Deriving and formalising safety and security requirements for control systems. In: SAFECOMP 2018. LNCS, vol. 11093, pp. 107–122. Springer (2018), https://doi.org/10.1007/978-3-319-99130-6_8
18. Vistbakka, I., Troubitsyna, E.: Towards a formal approach to analysing security of safety-critical systems. In: EDCC 2018. pp. 182–189. IEEE Computer Society (2018), https://doi.org/10.1109/EDCC.2018.00040
19. Vistbakka, I., Troubitsyna, E., Kuismin, T., Latvala, T.: Co-engineering safety and security in industrial control systems: A formal outlook. In: SERENE 2017. LNCS, vol. 10479, pp. 96–114. Springer (2017), https://doi.org/10.1007/978-3-319-65948-0_7
20. Young, W., Leveson, N.G.: An integrated approach to safety and security based on systems theory. Commun. ACM 57(2), 31–35 (2014), https://doi.org/10.1145/2556938