Analysing the Impact of Security Attacks on Safety Using SysML and Event-B*

Ehsan Poorhadi[1], Elena Troubitsyna[1], and György Dán[1]

KTH – Royal Institute of Technology, Stockholm, Sweden
{poorhadi,elenatro,gyuri}@kth.se

Abstract. Safety-critical control systems increasingly rely on networking technologies, which makes these systems vulnerable to cyber attacks that can potentially jeopardise system safety. To achieve safe- and secure-by-construction development, the designers should analyse the impact of security attacks already at the modelling stage. Since SysML is often used for modelling safety-critical systems, in this paper, we propose to integrate modelling in SysML and Event-B to enable reasoning about safety-security interactions at system modelling stage. Our approach combines the benefits of graphical modelling in SysML with the mathematical rigor of Event-B to visualise and formalise the analysis of the impact of security attacks on system safety.

Keywords: Safety-security interactions · Integrated approach · Formal specification and verification · Graphical modelling.

1 Introduction

Safety-critical control systems are increasingly relying on networking technologies. The resulting Networked Control Systems (NCS) inherently become the target of cyber attacks that can potentially jeopardize system safety. The interactions between safety and security are often complex, and thus, they should be analysed systematically and rigorously already in the modeling stage of system design.

SysML [9] is a graphical modeling language that is widely used for modelling safety-critical control systems. It allows the designers to describe the system architecture and the dynamic behaviour using convenient diagrammatic notation. SysML extensions addressing security have also been proposed [12]. However, while SysML facilitates a multi-view modelling of system behaviour, it should be combined with formal verification to ensure that the impact of cyber attacks on safety can be thoroughly analysed. Such an analysis should identify the conditions under which safety cannot be guaranteed and provide the input for defining appropriate security control mechanisms.

In this paper, we focus on analysing tampering and deletion attacks. We propose an integrated approach that combines graphical modelling in SysML with

* Supported by Trafikverket, Sweden.

formal specification and verification in Event-B [11]. Event-B is a state-based rigorous framework for correct-by-construction development of reactive systems. The framework has a powerful automated tool support – the Rodin platform [10]. Rodin provides an integrated modelling environment for specification and proof-based verification of complex systems.

In this paper, we use a subset of SysML diagrams to model the behaviour of a NCS. We define the main requirements that SysML diagrams should fulfil to enable their translation into Event-B, and an explicit analysis of safety-security interactions. Such requirements are formalised as consistency rules that should be verified to ensure correct translation into Event-B. The translation is integrated into the correct-by-construction refinement chain, i.e., at each refinement step, certain aspects of system behaviour represented in SysML are formally specified in Event-B. The resulting Event-B specification models the impact of attacks and identifies safety requirements, formalised as model invariants, which cannot be preserved. Such a specification serves as a basis for the consecutive security analysis aiming at defining security control mechanisms and prioritizing their implementation.

The rest of the paper is structured as follows. In Section 2 we define a generic architecture of NCS and define tampering and deletion attacks in this context. In Section 3, we define SysML diagrams used for modelling NCS. In Section 4, we briefly overview modelling in Event-B. In Section 5, we present our approach to translating SysML into Event-B and the reasoning about safety-security interactions, and in Section 6 we present the impact analysis of cyber attacks. In Section 7 we overview the related work and Section 8 concludes the paper.

2 Generic architecture of networked control system

In this section, we define a generic architecture of a NCS and discuss the impact of cyber attacks on safety. The generic architecture is shown in Figure 1. We assume, without loss of generality, that the system is responsible for maintaining some critical parameter P within a safe interval,

$$P_{min} \leq P \leq P_{max}.$$

The system behavior is cyclic. Each cycle executes a control loop that starts from the sensor measuring the parameter P and sending a message $sensor_{out}$ to the controller. The controller computes a command $cont_{out}$ and sends it to the actuator. The computation is based on a control algorithm and on the input $cont_{in}$ received from the sensor. Finally, the actuator changes its state according to the command $actuator_{in}$ that it receives from the controller. The actuator's state affects the value of the critical parameter P.

In this paper, we consider fail-safe systems, i.e., systems that can be put in a safe non-operational state if there is risk of a safety hazard. To keep the actual physical value of the critical parameter within the safe interval, we take into account an imprecision $\Delta \in \mathbb{N}$. Hence, the safety property can be defined as

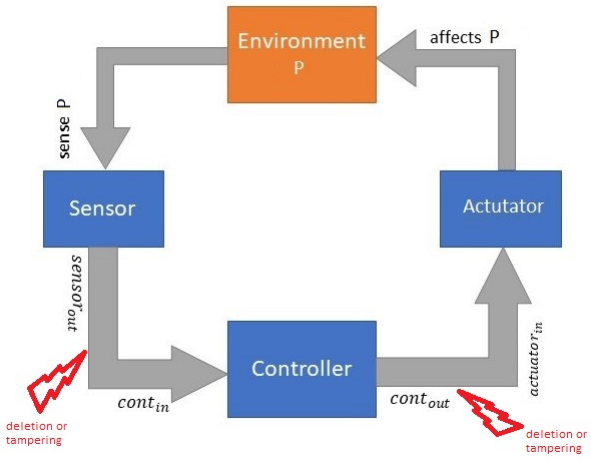

Fig. 1. Architecture of an NCS.

$$Safety: \quad P_{min} - \Delta \leq P \leq P_{max} + \Delta \quad \vee \quad FailSafe = TRUE, \tag{1}$$

In our generic architecture of an NCS there are wireless or wired communication channels between sensor-controller and controller-actuator. These channels can be attacked. In this paper, we study Man-in-the-middle attacks resulting in tampering or deletion of messages sent over the communication channels. These attacks on the sensor-controller channel would result in the controller receiving an incorrect sensor reading or no reading at all. As a consequence, the controller could compute an incorrect, possibly hazardous control command. An attack on the controller-actuator channel would result in the direct setting of the actuator to a dangerous state. We can formalize tampering attacks as

$$sensor_{out} \neq cont_{in} \quad \vee \quad cont_{out} \neq actuator_{in}, \tag{2}$$

i.e., the sent and the received messages are different. The deletion attack can be formalized as

$$cont_{in} = null \quad \vee \quad actuator_{in} = null, \tag{3}$$

where *null* represents an empty message. The attacker can combine deletion and tampering attacks on both channels to create a powerful attack scenario that can bypass defense mechanisms and violate safety.

To systematically analyze the impact of cyberattacks on safety, in this paper we propose an approach that combines graphical modeling in SysML with formal

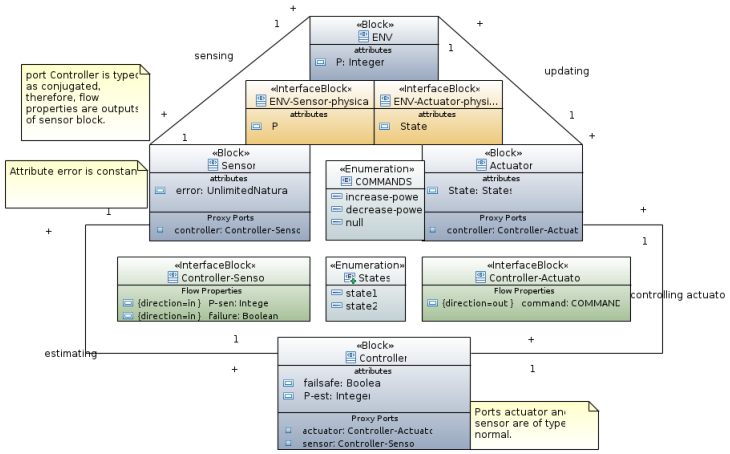

Fig. 2. A BDD for a generic NCS.

specification and formal verification in Event-B. In the next section, we will describe how to model an NCS in SysML in such a way that cyber attacks could be explicitly represented in the consequent translation into Event-B specifications.

3 SysML representation of NCSs

System modeling language (SysML)[9] is a general-purpose modeling language for system engineering applications. Different aspects of system architecture and dynamic behaviour can be modelled using different subsets of nine SysML diagrams. In our work, this subset includes the block definition diagram (BDD), the state machine diagram (SMD), and the sequence diagram (SD).

To represent the system architecture, we use BDD, which allows us to model system components and their interfaces. We use the SMD to represent the internal dynamic behavior of each component. Finally, we use the SD to model interactions between the components and represent communication protocols. Next we present our approach to SysML modelling of NCS that facilitates subsequent analysis of safety-security interactions in Event-B.

Block definition diagram To represent the architecture of an NCS, we create the BDD. The main modeling primitive of BDD is called a *block*. To explicitly represent cyber attacks, we require that each element of our architecture, including the communication channel, is represented as a corresponding instance of a block. Figure 2 shows the BDD for our generic NCS created according to this principle.

The blocks visualize internal constants and variables of components as typed *attributes* of blocks. For our sensor block, we define the constant value *error*

to show the maximum sensor imprecision. In the actuator block, we define an attribute to represent the state of the actuator. In the controller block, we define two attributes $failsafe$ and P_{est}. The attribute $failsafe$ models whether the system is put in the failsafe state. The attribute P_{est} shows the estimation of the critical parameter P.

The interface between a pair of components that communicates with each other is represented by an *interface block*. The interface block defines the data flows, i.e., describes the messages, which are exchanged between two components as directed *flow properties*. In Figure 2, the interface between the controller and sensor named *Controller-Sensor* has two flow properties *P-sen* and $failure$. For the interface between controller and actuator, we introduce a flow property *command* that shows the command that the controller sends to the actuator. The direction of a flow property is either *in* or *out*. Figure 2 shows that the direction of both flow properties in the interface block between the sensor and controller is *in*.

To specify the inputs and outputs of a component, a block uses *ports*, which are typed by an interface block. A port is typed either *normal* or *conjugated*. If a port is normal, it is used to send or receive the flow properties according to their directions defined in the interface block. Otherwise, the direction of the flow properties are interpreted as inverse. In Figure 2, the sensor port of the controller is typed with the *Controller-Sensor* interface as *normal* while the controller port of the Sensor block is typed with the same interface block but as *conjugated*. It implies that the flow properties $P - sen$ and $failure$ are outputs and inputs of the sensor and controller, respectively. In this work, we use integers, Boolean, or an enumeration set as the type flow properties and attributes.

In safety-critical system, safety requirements are often defined in terms of the values of a certain physical parameter. Hence, we also introduce the blocks representing the physical environment and corresponding interface blocks to represent, for example, that the sensor estimates the value of a critical physical parameter and the actuator affects its value. In our BDD diagram shown in Figure 2, it is represented by yellow interface blocks. While BDD allows us to define system components and identify interfaces between them, the actual dynamic interactions between the components is modeled using the SD.

Sequence diagram A SD is used to show the interaction between components. The diagram represents the contents and the order of messages. Moreover, the SD can depict complex interactions using interaction *fragments*.

In our modelling of an NCS, we use *loop fragments* to represent the control loop. Our lifelines represent the main components of our generic NCS: the sensor, controller, and actuator as shown in Figure 3. The lifelines correspond to the instances of the blocks in the BDD shown in Figure 2. Also, the parameters of the messages should comply with be the flow properties defined in the corresponding interface block. For example, the messages $Failure$ and $Measurement$ carry the flow properties of the *controller-sensor* interface block.

The *alt fragments* show conditional interactions. We also use the alt fragment to depict the effect the deletion and the tampering attacks on a specific message.

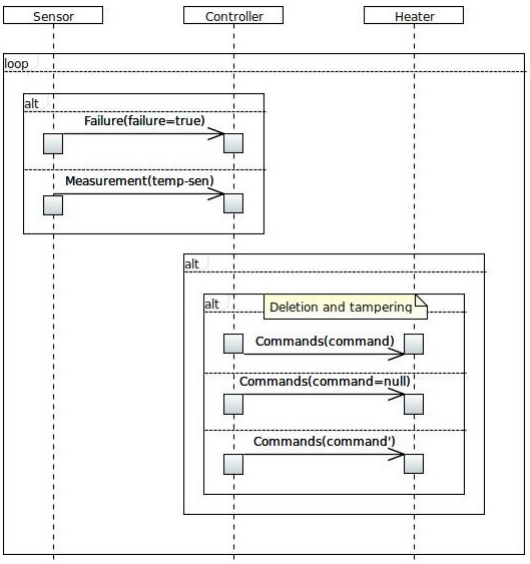

Fig. 3. An example of the SD for an NCS

For instance, consider the message *command* in Figure 3, which could be the target of deletion or tampering attacks. To represent this, we define an alt fragment that has three alternatives. The first alternative represents the absence of attacks. The second one shows the deletion attack by defining that the parameter command is equal to *null*. Finally, the last one represents the tampering attack. Here we use prime ($'$) to show the injected parameters.

State machine diagram A State Machine Diagram (SMD) is used to represent the internal behavior of system components. It models both the component functional logic and communication protocols. We create an SMD for each block in the BDD and rely on the attributes of the block to model component behaviour. We use the fact that the component (block) can read (write) the flow properties it receives (sends) from the block ports. Figure 4 shows an example of SMD for the block actuator. We require that the transitions defined in SMD are labeled and guarded.

Labels represent the messages that the corresponding block sends or receives. We use labels $r.M$ and $s.M$ for receiving and sending message M, respectively. If a transition is labeled with a message then the message must be received or sent upon triggering the transition.

We also introduce two extra labels *env* and *start* that can be used in SMDs of the blocks *Actuator* and ENV, respectively. If a transition is labeled with *env*, it implies that as a result of the transition, the actuator updates its states according to the commands. If a transition is labeled with *start*, it implies that, as a result of the transition, the critical parameter block ENV is updated. Figure

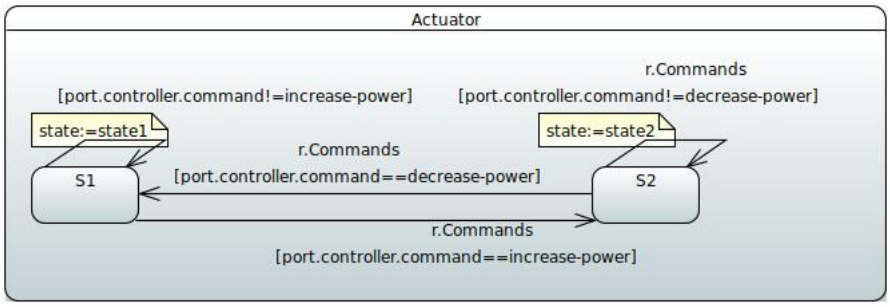

Fig. 4. An example of SMD for the actuator. The expressions in brackets show the guard of transitions.

4 shows four transitions that may be triggered when the message *Commands* is received.

The transitions may have some guard designating that the transition can occur only if the guard is evaluated true. The guards are the expressions defined using the attributes of the block and flow properties that the block receives. In the latter case, the transition should be labeled with a receiving message that carries the flow properties. SysML allows us to represent opaque behavior expressed in different languages. We describe the opaque behavior expressed in C to specify the guards. In Figures 4, the guards of transitions are shown in brackets.

The effect of the transition is described using a piece of code containing the following statements: assignment, *if* and a random choice from a given set. The code could assign to the attributes of the block or the output flow properties of the block. In the latter case the transition must be labeled with a sending message that carries the flow properties. Figure 4 shows the effect of two transitions updating the state of the actuator. Note that the behavior is specified using opaque behavior based on C, but in the figure, it is depicted using a comment box.

4 Modelling and Refinement in Event-B

Event-B is a state-based formal modelling approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving. In Event-B, a system model is specified using the notion of an *abstract state machine* [11]. An abstract state machine encapsulates the model state represented as a collection of variables and defines operations on the state, i.e., it describes the *behaviour* of the modelled system. Usually, a machine has an accompanying component, called *context*, which may include user-defined carrier sets, constants and their properties given as a list of model axioms. In Event-B, the model variables are strongly typed by the constraining predicates. These

predicates and the other important properties that must be preserved by the model constitute model *invariants*.

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, the *guard* G_e is a predicate over the local variables of the event and the state variables of the system. The body of the event is defined by the next-state relation R_e. In Event-B, R_e is defined by a *multiple* (possibly nondeterministic) assignment over the system variables. The guard defines the conditions under which the assignment can be performed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution non-deterministically.

Event-B employs a top-down refinement-based approach to system development. A development starts from an abstract system specification that non-deterministically models the most essential functional requirements. In a sequence of refinement steps, we gradually reduce non-determinism 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 so called *gluing invariant* as a part of the invariant of the refined machine. The gluing invariant defines the relationship between the abstract and concrete variables.

The consistency of Event-B models, i.e., verification of well-formedness and invariant preservation as well as correctness of refinement steps, is formally demonstrated by discharging the relevant proof obligations generated by the Rodin platform [10]. Rodin also provides an automated tool support for proving.

5 From SysML to Event-B: Translation Methodology

Our translation of SysML diagrams into Event-B aims at achieving two simultaneous goals. On the one hand, as the first step of translation, we check correctness of the SysML diagrams and we verify their consistency. On the other hand, we rely on formalization in Event-B to analyze the impact of cyber attacks on system safety. In this paper, due to lack of space, we only briefly outline our translation methodology and describe formal modelling of safety and security interactions in details.

Since BDD specifies the system´s static structure, it defines the name-space of all elements to be used in system modelling. In our translation, BDD serves as the main mechanism to check consistency between models. For example, when we are defining the SD, we can verify that messages coincide with the interface blocks. For SMD, we check whether the block owns the attributes and flow properties that are used in SMD, and the assignments comply their types. We also check the deadlock freeness of the SMDs. To check consistency of the SMD with respect to the SD, we check whether the transition labels correspond to the messages that the block sends or receives in the SD.

CONTEXT correctness

CONSTANTS

Measurement

P_sen

AXIOMS

axm1: $Measurement = \{P\_sen\}$

axm8: ⟨theorem⟩
$\forall fp, port \cdot fp \in Measurement \land port \in Interface(sensor \mapsto controller) \land PortOwner(port) = sensor \Rightarrow ($
$(Direction(fp) = out \Rightarrow conjugated(port) = FALSE) \land (Direction(fp) = in \Rightarrow conjugated(port) = TRUE)$
$)$

axm9: ⟨theorem⟩
$\forall fp, port \cdot fp \in Measurement \land port \in Interface(sensor \mapsto controller) \land PortOwner(port) = controller \Rightarrow ($
$(Direction(fp) = out \Rightarrow conjugated(port) = TRUE) \land (Direction(fp) = in \Rightarrow conjugated(port) = FALSE)$
$)$

END

Fig. 5. An excerpt from the context in which the correctness of SD will be checked with respect to BDD. As can be seen theorems 8 and 9 show that the direction of message is consistent with the direction of flow properties that it carries.

In our translation methodology, the consistency rules are defined as a separate context of the Event-B model. Such an approach mimics defining a meta-model as a part of formal modelling. The verification conditions are formulated as context theorems. If the proof of some theorem fails then an inconsistency between the models is detected. Upon detecting an inconsistency, the designer needs to change the SysML diagrams and perform the consistency checks again. Figure 5 shows an example of the context with the defined consistency rule.

When the consistency of the SysML model has been established, we shift the focus to translating SMDs and SD, which is incorporated into the Event-B refinement process. Once the refinement process is completed, the SysML models become fully formalised in Event-B and the detailed definition of the impact of cyber attacks on safety emerges via failed proofs of certain safety invariants.

Abstract specification The first machine (the abstract specification) is the translation of SD that focuses on modelling sending and receiving messages. While translating SD, we specify the sets representing each type of message. The name of the parameters of messages become the elements of the sets. For instance,

$$Measurement = \{P_{sen}\}.$$

To model sending and receiving messages, we define the set *Phases*,

$$Phases = \{s_{Failure}, r_{Failure}, s_{Measurement}, r_{Measurement}, s_{Commands}, r_{Commands}, env, start\}.$$

For each message, we define the "sending-receiving" order. In addition, *Phases* has elements designating the environment involving and the start of the control cycle.

To model the order of messages according to their position in the SD, we define the guards of the events ensuring that the previous messages have been sent before sending a message down in the lifeline becomes enabled.

To model tampering and deletion attacks on a message $M(par_1, ..., par_t)$, we define two variables,

$$send_M : \{par_1, ..., par_t\} \rightarrow \mathbf{Z}, \quad receive_M : \{par_1, ..., par_t\} \rightarrow \mathbf{Z},$$

which represent the values of the parameters of the messages after sending and receiving message M. If the attacker decides to perform the deletion attack, we set function $receive_M(par_i) = null$, where $null$ is an abstract constant representing an empty message. In case of the tampering attack, we choose a non-deterministic value v_i other than $send_M(par_i)$, and define $receive_M(par_i) = v_i$. Finally, in the absence of attacks, we define $receive_M = send_M$. To model sending and receiving a message, we define two events for each message correspondingly.

The final step in the SD translation is to define an event for each physical interface. This event only changes the phase of interactions. The event corresponding to the actuator-environment interface changes the phase from r_M to env in which M is the last message in the control loop. This event models the actuator's behaviour – changing its state according to the controller's commands.

The event corresponding to the environment-sensor interface changes the value of the *phase* from *env* to *start*. This event models the end of the environment evolution resulting in the update of the critical parameter. Figure 6 shows an excerpt from the abstract specification obtained as a result of SD translation. As can be seen, to ensure that the message *Command* will be sent after either $Measurement$ or $Failure$, we add guard $grd2$ to event *send-Command*. To model the attacks on the message *Command*, we add guards $grd2$, $grd3$, and $grd4$ to event *receive-Command*.

Translation-Driven Refinement Our subsequent refinement steps aim at capturing the details of SMDs modelling the behavour of system components. Our translation and the corresponding refinement process has the following order: representing the behaviour of the environment (ENV), actuator, controller, and finally, sensor.

We start the translation of the SMD with defining the set *States* containing all states as its elements. To model the current state, we define the variable $state_B \in States$, where B is the name of the block.

For each transition in the corresponding SMD, we define an event that is enabled if the current state is equal to the source of the transition. As result of triggering the event, the current state is changed to the destination state of the transition. If a transition is labeled with sending or receiving a message M ($s.M$ *or* $(r.M)$) then the corresponding event refines the event that models sending or receiving M. If a transition in the actuator's SMD (ENV's SMD) is labeled with *env* (*start*) then the corresponding event refines the abstract event defined

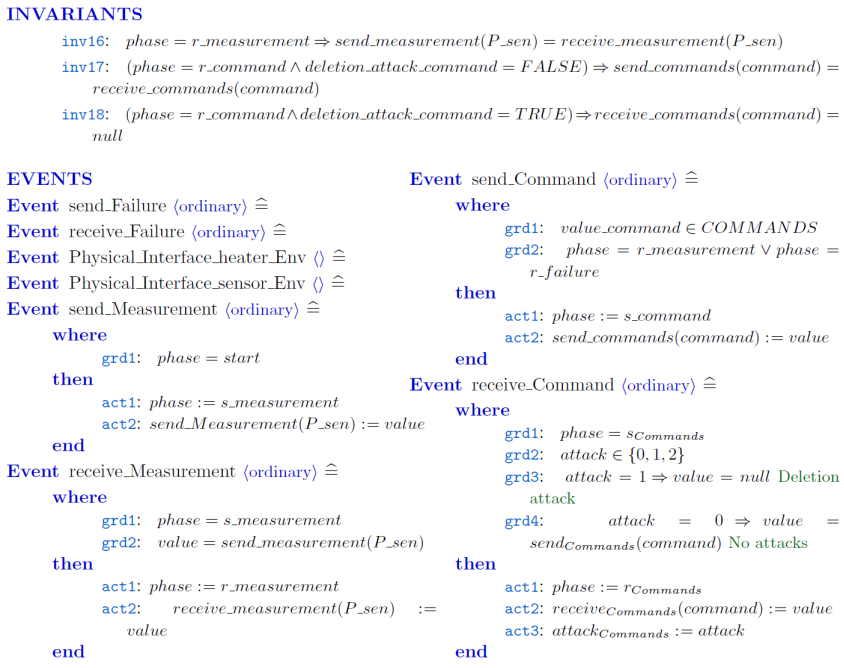

Fig. 6. An excerpt of abstract specification which is the translation of SD.

for the Actuator-ENV (ENV-Sensor) physical interface. To specify the events, we translate the guard of the transition into the guard of the corresponding event. The effect of the transition is translated into the actions of the corresponding event.

To translate *guards* and *behaviours*, we introduce a variable for each attribute (that is not a constant) of the relevant block. The dictionary $CtoB$ presented in Table 1 shows how guards and effects of the transitions are translated in Event-B. For translating assignments to an attribute x, we first create a parameter par_x and add the action $x := par_x$. Then for an assignment $x := e$, we add $par_x = CtoB(e)$ to the guard of the event. Note that, in order to avoid deadlock, each attribute must be assigned once per transition. If an assignment is inside of an *if statement*, we add,

$$par_{if} = TRUE \Rightarrow par_x = CtoB(e),$$

to the guard instead of $par_x = CtoB(e)$. A guard g is also translated using $CtoB$ dictionary, however, if the guard is inside of a if statement, then we apply the similar technique. Figure 7 shows the translation of the SMD shown in Figure 4.

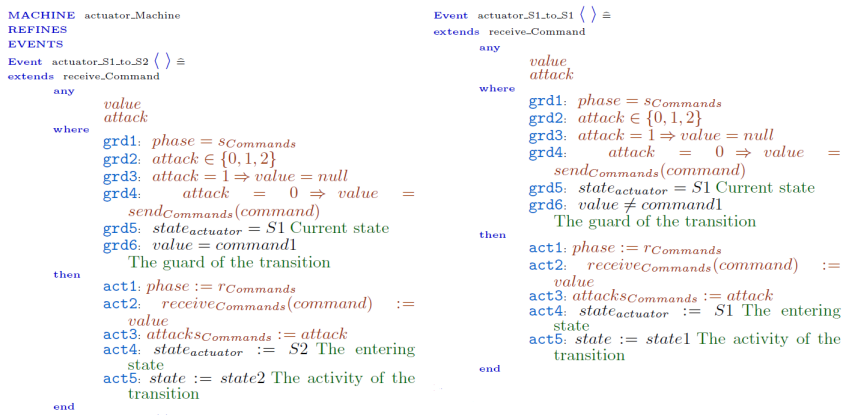

Fig. 7. Translation of actuator's SMD into Event-B. Only two out of four events are shown. Red lines are inherited from abstract event in previous machine.

Table 1. SysMLtoB dictionary.

C	Event-B
&&, $\|\|$	$\wedge, \vee$
==, <=, >=, <, >, =!	$=, \leq, \geq, >, <, \neq$
$+, -, *, /$	$+, -, \times, \div$
$if\ c\ \{\ \ \}$	**Any** par_{if} **where** $par_{if} = TRUE \Leftrightarrow CtoB(c)$
$rand(1, n)$	**Any** par **where** $par \in 1..n$

6 Analysing the Impact of Cyber Attacks on Safety

In this section, we discuss our approach to verifying the impact of cyber attacks on safety within the SysML translation-driven refinement process. In our approach, the impact of cyber attacks on safety is analysed by checking whether the safety property 1 remains provable, i.e., if after the detailed representation of the effect of the attack on the system behaviour safety invariants remain provable then the cyber attack does not have a safety implication, and vice versa – if the invariants become violated then the cyber attack has safety implications.

Once the SMD diagram is translated into Event-B, we obtain a detailed specification of the corresponding system component and can analyse the impact of a cyber attack on its behaviour. Let $safety'$ be the first term of the disjunction defined in property 1, i.e., safety should be guaranteed after updating the physical parameters at the end of the control loop. As an example, $safety'$ can be defined either as

$$P_{min} - \Delta \leq P + x \leq P_{max} + \Delta,$$

or as

$$P_{min} - \Delta \leq P - x \leq P_{max} + \Delta,$$

for some positive x depending on the state of the actuator ($state1$ or $state2$). We define the following invariants postulating that the actuator enforces safety by preventing the critical parameter from passing the safety margin when it updates its state as follows:

$$(phase = env \ \wedge state = state1) \Rightarrow P_{min} - \Delta \leq P + x \leq P_{max} + \Delta,$$

$$(phase = env \ \wedge state = state2) \Rightarrow P_{min} - \Delta \leq P - x \leq P_{max} + \Delta.$$

The proof of the above invariants requires demonstrating that the actuator should be able to receive the correct command from the controller irrespective whether the attack is carried out or not. This implies that we should guarantee that the controller logic is correct and the attacker actions cannot put the actuator in a state that causes violation of above invariants. There requirements are modelled by the guards of the events representing sending and receiving *Commands*. Later, in the next refinement in which we translate the controller's SMD, we remove these guards.

In the next refinement steps, when the controller's SMD is translated, we change the requirements from the guard of the events to be theorems. At this point, we need to prove that the logic of the controller implies the theorems. The key invariants that conclude the theorems are,

$$phase = s_{Commands} \Rightarrow failsafe = FALSE,$$

$$phase = s_{Commands} \ \wedge \ send_{Command}(command) = increasePower \Rightarrow$$

$$P_{min} - \Delta \leq P + x \leq P_{max} + \Delta,$$

and

$$phase = s_{Commands} \ \wedge \ send_{Command}(command) = decreasePower \Rightarrow$$

$$P_{min} - \Delta \leq P - x \leq P_{max} + \Delta,$$

Proving the invariants requires that security control mechanisms detect attacks on both sensor-controller and controller-actuator channels because they lead to safety violation. In this case, the controller sets $falisafe = TRUE$, and we never reach a state of the model in which $phase = s_{Command}$. Also, depending on the details of the controller logic, we may need to introduce assumptions about the sensor´s imprecision. These assumptions can be added to the guard of event in which message $Measurement$ is received. In the next refinement step resulting in adding the model of the sensor, we prove the assumption about the sensor imprecision.

Finally, upon translating all SMDs, we can define the safety property 1 as an invariant of the final model and try to prove it. At this point, the prove should be discharged without introducing any further assumptions.

7 Related work

Several previous works proposed different modelling approaches to represent the impact of security attacks on safety within Event-B framework [1] [2]. The authors explored different refinement strategies as well as the use of HAZOP for requirements elicitation [3]. While our approach adopts the idea of an incremental unfolding system architecture via refinement, we rely on SysML models as an intermediate step between the informal requirements description and formal specification. We were encouraged by railway industry to integrate SysML into our modelling to help system engineers to understand both formal models and security analysis results. The work on formal modelling of cyberattacks in the railway domain is reported in our previous work [4]. In this paper, we have extended the modelling methodology proposed in [4], on the one hand, to support an integration between SysML and Event-B for generic NCS, and, on the other hand, to include modelling of deletion and tampering attacks on both channels of NCS.

The work on integrating general purpose graphical modelling and formal specification has been carried by Snook et al. [5]. Currently, there is also an UML-B plugin is available for the Rodin platform that enables an automatic translation of UML-B into Event-B specification [6]. We have adopted the similar techniques in translating state machine diagrams but focused on explicit modelling and verification of safety-security interactions.

UML-B has been used in some works to analyze the safety of NCS. For example, in [7], UML-B is used to model hybrid ERTMS level 3. However, our work includes a broader set of SysML diagrams and allows to express a richer set of properties.

Quamara et al. [8] proposed a multi-layer model-based engineering approach addressing safety-security interactions. They also rely on graphical and formal modelling but focus on identifying conflicts between safety and security requirements.

8 Conclusion

In this work, we proposed an integrated approach to modelling the impact of cyber-security attacks on safety of networked control systems. We have defined the principles of modelling networked control systems in SysML to enable an explicit reasoning about safety-security interactions. We also introduced a methodology for translating SysML BDDs, SD and SMDs into Event-B. Our methodology supports the verification of consistency of SysML diagrams as well as enables a rigorous analysis of the impact of security on safety.

Our SysML translation into Event-B has been incorporated into the correct-by-construction development process. It allowed us to explicitly analyse the impact of security attacks on the behaviour of each component as well as the overall system safety. The resultant model explicitly demonstrates which safety invariants become violated as the result of the attacks. Pro-B model checker can be used to construct the concrete attack scenarios.

The work on the integration of SysML and Event-B for modelling safety-security interactions in networked control systems has been motivated by our cooperation with railway industry. Since SysML is widely used for system modelling, we believe that the proposed integration approach can facilitate understanding of formal models and the results of modelling the security attacks by the industrial engineers.

As a future work, we are planning to investigate modelling of the other types of attacks, such as injection, reordering and delaying. It would be also interesting to work on representing the results of formal modelling in SysML, e.g. to visualise the attack scenarios. There is also an ongoing work on the automation of the proposed approach.

References

1. Troubitsyna, E., Laibinis, E., Pereverzeva, I., Kuismin, T., Ilic, D., Latvala, T. Towards Security-Explicit Formal Modelling of Safety-Critical Systems. Safecomp 2016, LNCS 9922, pp. 213-225, 2016. https://doi.org/10.1007/978-3-319-45477-117
2. Vistbakka, I., Troubitsyna, E., Kuismin, T., Latvala, T. (2017). Co-engineering Safety and Security in Industrial Control Systems: A Formal Outlook. SERENE, LNCS 10479, pp. 96-114, 2017. https://doi.org/10.1007/978-3-319-65948-07
3. Troubitsyna,E., Vistbakka, I. Deriving and Formalising Safety and Security Requirements for Control Systems. SAFECOMP 2018. LNCS 11093, pp.107-122, Springer.
4. Poorhadi, E., Troubitsyna, E., Dán, G. (2021). Formal Modelling of the Impact of Cyber Attacks on Railway Safety. In: SAFECOMP 2021 Workshops. SAFECOMP 2021. LNCS, vol. 12853. https://doi.org/10.1007/978-3-030-83906-2_9
5. Snook, C., Butler, M. 2006. UML-B: Formal modeling and design aided by UML. ACM Trans. Softw. Eng. Methodol, pp. 92–122, 2006 https://doi.org/10.1145/1125808.1125811
6. UML-B Homepage, https://www.uml-b.org/.
7. Dghaym, D., Dalvandi, M., Poppleton, M., Snook, C. 2020. Formalising the Hybrid ERTMS Level 3 specification in iUML-B and Event-B. Int. J. Softw. Tools Technol. Transf. 22, 3 (Jun 2020), 297–313.
8. Quamara, M., Pedroza, G., Hamid, B. Multi-layered Model-based Design Approach towards System Safety and Security Co-engineering. 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 274-283, https://doi.org/10.1109/MODELS-C53483.2021.00048
9. SysML Homepage, https://sysml.org/.
10. The RODIN platform, http://rodin-b-sharp.sourceforge.net/.
11. Abrial,J. Extending B without Changing it (for Developing Distributed Systems). Proceedings of 1st Conference on the B Method, pp.169-191, Springer-Verlag, November 1996, Nantes, France
12. Lemaire, L., Lapon, J., Decker, B., Naessens, V. 2014. A SysML Extension for Security Analysis of Industrial Control Systems. In Proceedings of the 2nd International Symposium on ICS and SCADA Cyber Security Research 2014 (ICS-CSR 2014). BCS, Swindon, GBR, 1–9. https://doi.org/10.14236/ewic/ics-csr2014.1