A Formal, Systematic Approach to STPA using Event-B Refinement and Proof

John Colley and Michael Butler

Electronics and Computer Science, University of Southampton

Southampton, UK

Abstract System-Theoretic Process Analysis (STPA) from Leveson is a technique for hazard analysis developed to identify more thoroughly the causal factors in complex safety-critical systems, including software design errors. Event-B is a proof-based modelling language and method that enables the development of specifications using a formal notion of refinement. We propose an approach to hazard analysis where system requirements are captured as monitored, controlled, mode and commanded phenomena and STPA is applied to the controlled phenomena to identify systematically the safety constraints. These are then represented formally in an Event-B specification which is amenable to formal refinement and proof.

1 Introduction

System-Theoretic Process Analysis (STPA), described in (Leveson 2012), is a technique for hazard analysis developed to identify more thoroughly the causal factors in complex safety-critical systems, including software design errors. STPA has been applied to a wide range of safety critical applications (Leveson 2012). Event-B (Abrial 2010) is a proof-based modelling language and method that enables the development of specifications using a formal notion of refinement. The Rodin platform (Abrial et al. 2010) is the Eclipse-based IDE that provides automated support for Event-B modelling, refinement and mathematical proof. The Event-B method has also been used in the deployment of safety critical systems for automotive and railway applications.

We propose an approach to hazard analysis where system requirements are captured as *monitored*, *controlled*, *mode* and *commanded* phenomena and STPA is applied to the controlled phenomena to identify systematically the safety constraints. These are then represented formally in an Event-B specification, which is amenable to formal refinement and proof.

In Section 2 we provide an overview of Event-B with particular attention paid to formal Event-B refinement.

In Section 3 we show how our proposed approach to hazard analysis can be applied, using a domestic washing machine case study, to derive systematically the safety constraints expressed in *natural language*.

In Section 4 we illustrate how the natural language safety constraints can be represented formally in Event-B.

In Section 5 we present a summary of our approach and the direction of our future work.

2 Event-B

In Event-B, an abstract model comprises a *machine* that specifies the high-level behaviour and a *context*, made up of sets, constants and their properties, that represents the type environment for the high-level machine. The machine is represented as a set of *state variables*, *v* and a set of events, *guarded atomic actions*, which modify the state. If more than one action is enabled, then one is chosen non-deterministically for *execution*, an observable transition on the state variables which must preserve an *invariant* on the variables, *I(v)*.

A more concrete representation of the machine may then be created which refines the abstract machine, and the abstract context may be extended to support the types required by the refinement. *Gluing invariants* are used to verify that the concrete machine is a correct *refinement*: any behaviour of the concrete machine must satisfy the abstract behaviour. Gluing invariants give rise to proof obligations for pairs of abstract and corresponding concrete events. Events may also have parameters, which take, non-deterministically, the values that will make the guards in which they are referenced true.

Event-B refinement allows a model to be built gradually (Abrial and Hallerstede 2006), starting with an abstract model and then introducing successive, more concrete refinements. Adding variables achieves spatial extension and adding events temporal extension. Events in the abstract model may be refined by one or more events in the concrete model. These concrete events can modify the state of new variables introduced in the refinement, but must preserve the behaviour with regard to the variables declared in the abstract model. New events may also be introduced in the refinement. These events are not allowed to assign values to abstract variables, but can assign values to new variables introduced in the refinement.

3 The washing machine case study

We use this case study to explore a systematic method for identifying both functional and safety requirements. We start with an overview of the washing machine system.

3.1 System overview

We are concerned with developing a master controller, which, on receiving a set of user settings from the control panel, will control the water drum system and agitator motor to comply with those user settings.

3.2 Discovering the functional requirements

We investigate the functional requirements using a method that identifies the system phenomena and then structures the functional requirements according to these phenomena (Yeganefard and Butler 2012). The phenomena that we shall explore are the *monitored p*henomena, *commanded* phenomena, *controlled* phenomena and *mode* phenomena.

3.3 Monitored phenomena

First we examine the phenomena that will be monitored by the washing machine controller.

3.3.1 Drum water level

The controller will receive the current level from the water level sensor.

3.3.2 Drum water temperature

The controller will receive the current temperature from the water temperature sensor.

3.3.3 Door position

The controller will receive from the door sensor whether the door is closed or open.

3.3.4 Vibration level

The controller will receive from the vibration sensor the level of vibration.

3.4 The commanded phenomena

These are the phenomena that are driven by the user through the washing machine control panel.

3.4.1 Water level setting

The controller will receive the water level setting from the control panel. In this case two settings are possible: half load and full load.

3.4.2 Cycle setting

The controller will receive the cycle setting identifier from the control panel and decode the cycle setting. The cycle setting consists of

- the mode sequence, for example: idle, wash, rinse, spin, rinse, spin, idle
- the mode duration: how long each mode will run
- the spin speed.

3.4.3 Water temperature setting

The controller will receive the water temperature setting from the control panel: 30, 40 or 60 degrees Celsius.

3.4.4 Start signal

The controller will receive the start signal from the control panel.

3.5 The controlled phenomena

These are the phenomena that are driven by the master controller.

3.5.1 Door lock

- The controller will lock the door at the start of the cycle.
- The controller will unlock the door at the end of the cycle.
- The door will remain locked during the cycle.

3.5.2 Agitator motor

- The controller directs the speed and rotation direction of the agitator motor.
- The agitator motor will be stationary when the door is unlocked.

3.5.3 Water control valves

The controller activates and de-activates the hot and cold water valves to meet the water level and temperature requirements.

3.5.4 Water drain pump

The controller activates the water drain pump to meet the water level requirements.

3.5.5 Heater

The controller activates and de-activates the heater to meet the temperature requirements.

3.6 The mode phenomena

The controller modes are idle, washing, rinsing and spinning.

3.7 Discovering the safety requirements

The following two quotations from (Leveson 2012) encapsulate the approach to safety analysis, developed by Leveson, which we use in the case study.

> Any controller – human or automated – needs a model of the process being controlled to control it effectively.
>
> Accidents can occur when the controller's process model does not match the state of the system being controlled and the controller issues unsafe commands.

Simply trying to make components more reliable does not in itself make a system safer. Safety is enhanced when the controller(s) respond to component failures in a way which ensures that the resulting hazards are correctly and safely managed.

Consider a potential hazard arising from the heater sub-system of the washing machine. The water could overheat dangerously if the controller cannot monitor water temperature properly. If the temperature sensor is faulty, the controller could switch off the heater if the value read from sensor is out of operating range. If, however, the sensor reports a value within the operating range but the actual value is *out of operating range*, how can the controller respond to this hazard? Sensor redundancy, with the introduction of a voting system in the controller, can decrease the probability that the hazard will not be detected. An alternative approach, however, is for the controller to predict the rise in water temperature and compare it with the reported rise.

The controller needs independent verification of the sensed values to detect failure. This can be provided by values from a different sensor or the controller can generate predicted values in the absence of other sources of data.

3.8 System-Theoretic Process Analysis

Leveson proposes a rigorous approach, System-Theoretic Process Analysis (STPA), which consists of the following three steps.

1. Identify potentially hazardous control actions.
2. Derive the safety constraints.
3. Determine how unsafe control actions could occur.

STPA has been used by the US Missile Defense Agency to characterize the residual safety risk of the ballistic missile defense system (Perreira et al. 2006). A simulator of the interceptor flight computer is used to predict the expected behaviour and therefore to detect a failure in the system.

In our method, we perform a systematic analysis of the *controlled phenomena* identified in the requirements analysis: the door lock, the heater, the water drain pump, the water control valves and the agitator motor.

3.8.1 The door sub-system

Consider a model of the controlled door sub-system as shown in Figure 1.

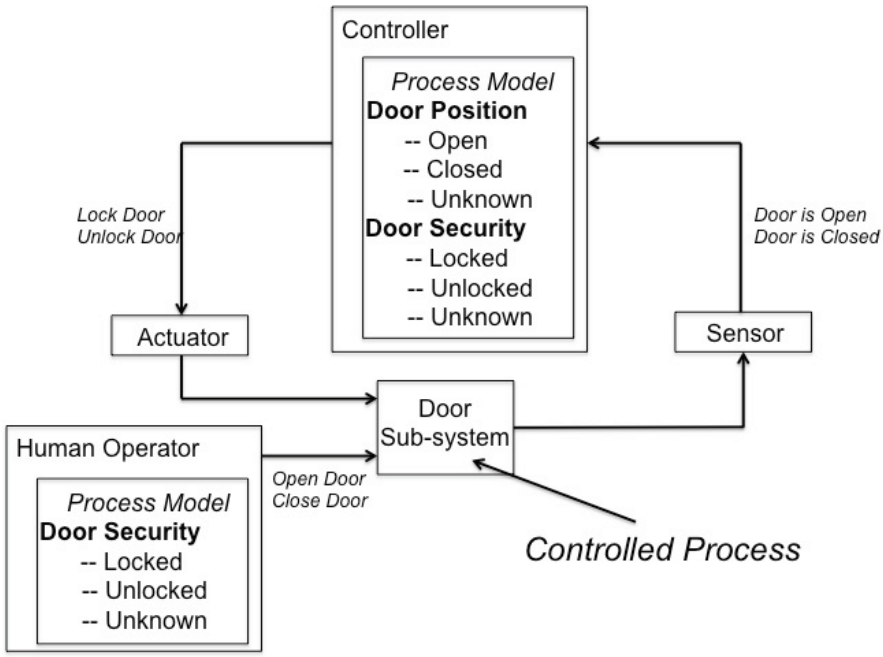

Fig.1. The controlled door sub-system

The main controller has a process model of the door sub-system. So also does the human operator. The operator can open or close the door directly. The controller uses an actuator to lock and unlock the door and a sensor to detect whether the door is open or closed.

Step 1: identifying potentially hazardous control actions

For each of the two controller actions, *Unlock Door* and *Lock Door*, we identify three potential causes of a hazard: *not providing* the action when it should, *providing* the action when it shouldn't and providing the action at the wrong time or in the wrong order. The results of the analysis are shown in Table 1.

Failing to unlock the door is inconvenient but not hazardous. Unlocking the door when the drum is filled is hazardous because the operator will be able to open the door inadvertently and release potentially very hot water. Unlocking the door *before* the drum has been fully drained is also hazardous.

Table 1. Door hazards

Controller action	Not providing causes hazard	Providing causes hazard	Wrong timing or order causes hazard
Unlock Door	Not hazardous	Operator can open door with drum filled	Water not fully drained
Lock Door	Operator can open door with drum filled	Not hazardous	Water starts filling before lock

Failing to lock the door when the drum is filled is hazardous, but locking the door when the drum is empty is not. Locking the door *after* the drum has started filling is hazardous.

Step 2: deriving the safety constraints

Three safety constraints can be derived from Table 1.

1. The door must always be locked when there is water in the drum.
2. An *Unlock Door* command must never be issued until the water is fully drained.
3. A *Lock Door* command must be issued before starting to fill the drum.

The first is an *invariant* of the system. The second and third are *guards* that prevent an operation occurring in an unsafe way. These natural language invariants and guards can then be represented formally in an Event-B model, as we shall show in detail in Section 4.

Step 3: determining how unsafe control actions could occur

We now revisit the controlled door sub-system to determine systematically the potential causes of unsafe actions as shown in Figure 2.

The hazard is that the door is open when there is water in the drum. The potential causes of this hazard are then represented on the diagram. The controller or the operator can have an inadequate or incorrect process model of the door subsystem, the requirements may not be fully specified or implemented and the operator may not be properly trained. The actuators and sensors may fail. These potential causes of unsafe actions can be used to both improve the design and inform the test plan.

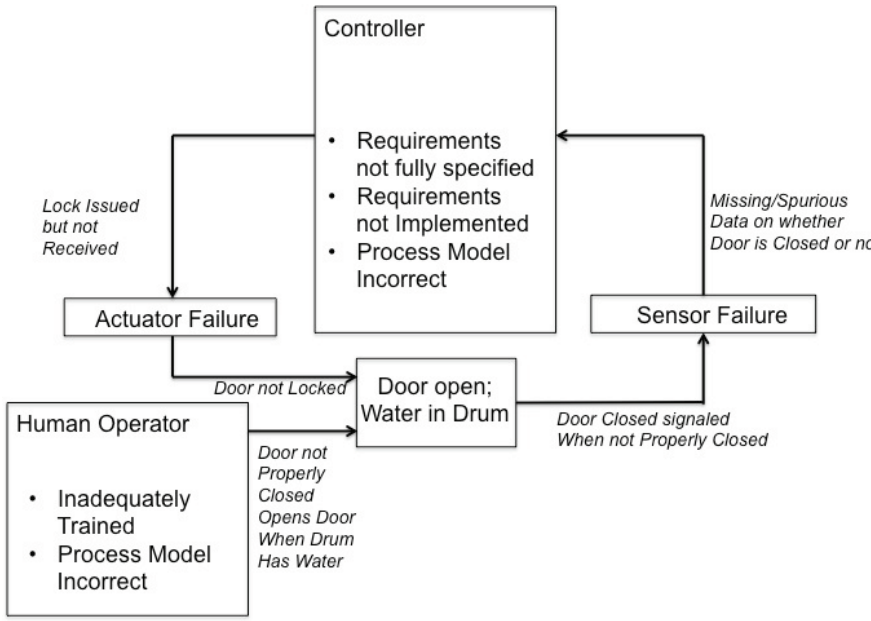

Fig.2. Potential causes of unsafe actions

4 Representing the safety constraints formally in Event-B

4.1 The Abstract Model: the Door Sub-system

To illustrate the method, we present first an abstract model of the washing machine door sub-system. We define an Event-B *context* as shown in Figure 3.

```
context LOCKSC

constants OPEN CLOSED LOCKED UNLOCKED

sets DoorPosition DoorState

axioms
  @axm1 partition(DoorPosition, {OPEN}, {CLOSED})
  @axm2 partition(DoorState, {LOCKED}, {UNLOCKED})
end
```

Fig.3. Door sub-system context

The *position* of the door can either be *OPEN* or *CLOSED* and the *state* of the door can either be *LOCKED* or *UNLOCKED*.

We then define an Event-B *machine*, which *sees* this context, as shown in Figures 4 and 5.

```
machine LOCKSM sees LOCKSC

variables dpos doorst

invariants
  @inv1 dpos ∈ DoorPosition
  @inv2 doorst ∈ DoorState
  @inv3 doorst = LOCKED ⇒ dpos = CLOSED

events
  event INITIALISATION
    then
      @act1 dpos ≔ OPEN
      @act2 doorst ≔ UNLOCKED
  end
.
.
```

Fig.4. Door sub-system machine initialisation

The variables *dpos* and *doorst* represent the door position, which is initialized to *OPEN*, and the door state, which is initialized to *LOCKED*. The *invariant @inv3* states that if the door is locked, then it must be closed.

The *events* of the machine are shown in Figure 5.

When the door is *open*, as indicated by the *guard @grd1* in the event *CloseDoor*, then the door can be *closed*. The guards of event *OpenDoor* indicate that the door can only be opened if it is *closed* and *unlocked*. The event *LockDoor* is only enabled if the door is *closed* and *unlocked*. The event *UnlockDoor* unlocks the door if it is locked.

The proof obligations for the machine are generated and discharged automatically by the Rodin tool. In particular, we have proved that the invariant *@inv3* is preserved for all possible *interleavings* of the events.

4.2 The refined model: introducing the drum sub-system

In the formal refinement of the abstract model, we first introduce the *drum state* in the context shown in Figure 6, which *extends* the abstract context.

The drum is either *EMPTY*, *FILLING*, *FILLED* or *EMPTYING*. The *refined* machine *sees* the extended context, introduces the variable *drumst* to represent the

state of the drum, refines the events of the abstract machine and introduces the events shown in Figure 7.

```
.
.
event CloseDoor
  where
    @grd1 dpos = OPEN
  then
    @act1 dpos ≔ CLOSED
 end

 event OpenDoor
  where
    @grd1 dpos = CLOSED
    @grd2 doorst = UNLOCKED
  then
    @act1 dpos ≔ OPEN
 end

 event LockDoor
  where
    @grd1 doorst = UNLOCKED
    @grd2 dpos = CLOSED
  then
    @act1 doorst ≔ LOCKED
 end

 event UnlockDoor
  where
    @grd1 doorst = LOCKED
  then
    @act1 doorst ≔ UNLOCKED
 end
end
```

Fig.5. Door sub-system machine events

```
context LOCKSE1 extends LOCKSC

constants EMPTY FILLING FILLED EMPTYING

sets DrumState

axioms
  @axm1 partition(DrumState, {EMPTY}, {FILLING}, {FILLED}, {EMPTYING})
end
```

Fig.6. Extended context

```
event FillDrum
  where
    @grd1 doorst = LOCKED
    @grd2 drumst = EMPTY
  then
    @act1 drumst ≔ FILLING
 end

 event Wash
  where
    @grd1 drumst = FILLING
  then
    @act1 drumst ≔ FILLED
 end

 event EmptyDrum
  where
    @grd1 drumst = FILLED
  then
    @act1 drumst ≔ EMPTYING
 end

 event Finish
  where
    @grd1 drumst = EMPTYING
  then
    @act1 drumst ≔ EMPTY
 end
```

Fig.7. New events in the refinement

The event *FillDrum* is only enabled if the door is *locked (@grd1)* and the drum is *empty (@grd2)*. The door can only be locked by the *LockDoor* event. These guards therefore fulfil the requirement of the safety constraint: '*A lock door command must be issued before starting to fill the drum*'.

To represent the safety constraint, '*The door must always be locked when there is water in the drum*', we introduce the invariant shown in Figure 8.

```
@inv2 drumst ≠ EMPTY ⇒ doorst = LOCKED
```

Fig.8. Safety constraint invariant

However, when we run the automatic prover, we find that this invariant cannot be proved for the *UnlockDoor* event. Inspecting the failing proof more closely, we see that the door can be unlocked while the drum is not *empty*. We must therefore *strengthen* the guards of the abstract event in this refinement by introducing the extra guard, *@grd2,* as shown in Figure 9.

This guard represents the safety constraint: '*An Unlock Door command must never be issued until the water is fully drained*.' When we re-run the prover, the invariant, *@inv2*, is now proved automatically. All three safety constraints derived during the safety analysis are now represented formally in the refined model.

1. The door must always be locked when there is water in the drum.
2. An *Unlock Door* command must never be issued until the water is fully drained.
3. A *Lock Door* command must be issued before starting to fill the drum.

```
event UnlockDoor refines UnlockDoor
  where
    @grd1 doorst = LOCKED
    @grd2 drumst = EMPTY
  then
    @act1 doorst ≔ UNLOCKED
end
```

Fig.9. Safety constraint guard

5 Summary and future work

We have presented an approach to hazard analysis where system requirements are captured as *monitored*, *controlled*, *mode* and *commanded* phenomena and STPA is applied to the controlled phenomena to identify systematically the safety constraints. These natural language constraints are then represented formally in an Event-B specification, which is amenable to formal refinement and proof. We have shown how the safety constraints are represented as either *invariants* or *guards* in the formal model. We build the model systematically using Event-B formal refinement. The Rodin environment automatically generates the required proof obligations, and the Rodin provers have been shown in this case study to discharge the proof obligations automatically. Where a proof obligation cannot be discharged, we have shown how the Rodin tool guides the user to improve the model.

We have illustrated our approach using the door lock phenomenon. Application of the method continues by analyzing and modelling in the same way the remaining *controlled phenomena*: the agitator motor, water control valves, water drain pump and water heater.

It is an important goal of our work to integrate Event-B based formal verification techniques into the overall system development flow. In particular, it is necessary to validate the specification against the original requirements. This cannot be achieved in an ad hoc manner. It is necessary to trace elements of the specification back to the requirements, and for this tool support is vital to ensure that there is a measurable way of ensuring that the requirements are covered by the specification. In future work, therefore, we shall integrate our approach with the requirements capture and tracing facility, ProR (Jastram 2010) that forms part of the Rodin platform. ProR provides a flexible and configurable environment to support

requirements engineering, within which we will integrate requirements analysis and safety analysis within the Rodin toolset and workflow.

Acknowledgments The research presented in this paper is funded by the FP7 ADVANCE (287563) project, Advanced Design and Verification Environment for Cyber-physical System Engineering, http://www.advance-ict.eu.

References

Abrial J-R (2010) Modeling in Event-B – System and Software Engineering. Cambridge University Press

Abrial J-R et al (2010) Rodin: an open toolset for modelling and reasoning in Event-B. STTT, 12(6):447–466

Abrial J-R, Hallerstede S (2006) Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamenta Informaticae, XXI

Jastram M (2010) ProR, an open source platform for requirements engineering based on RIF. SEISCONF

Leveson N (2012) Engineering a safer world: Systems thinking applied to safety. MIT Press (MA)

Perreira S et al (2006) A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system. Technical report, DTIC Document

Yeganefard S, Butler M (2012) Control systems: Phenomena and structuring functional requirement documents, 17th International Conference on Engineering of Complex Computer Systems (ICECCS)