D.5.1 - ADVANCE PROCESS INTEGRATION I

ADVANCE

Grant Agreement: *287563*
Date: *25th September, 2012*
Pages: *31*
Status: *Final*
Access: *Access List*
Reference: *N/A*
Issue: *1.0*

Partners / Clients:

FP7 Framework Programme ***European Union***

Consortium Members:

University of Southampton ***Critical Software Technologies*** ***Alstom Transport*** ***Systerel*** ***Heinrich Heine Universität Düsseldorf***

Contributors:

Lukas Ladenberger
Stefan Hallerstede
Michael Jastram
John Colley

Reviewers:

Laurent Voisin

Contents

Chapter 1

Preface

WP5 D5.1 deliverable definition: ADVANCE Process Integration I.

This deliverable reports on the initial work on combining formal modelling with requirements analysis and safety analysis and measures progress of these activities against plan. The emphasis is to review the development of the methods that will need to be in place to enable requirements and safety analysis to be linked to formal models. The progress on tool development is then to be assessed to ensure that this development will meet the methodological requirements.

In chapter 2 we present the motivation for the work in the two primary areas: Requirements Analysis and Safety Analysis.

In chapter 3 we deal with Requirements Analysis in detail, describing an incremental approach to requirements modelling and validation that incorporates formal and informal reasoning and the tool support developed to support this approach.

In chapter 4 we describe the Safety Analysis method we will adopt for ADVANCE and how we plan to integrate this approach with the Requirements Analysis method and tooling described in chapter 3.

In chapter 5 we summarise the work done so far for this workpackage and the focus of future work.

Chapter 2

Process Integration Objectives

2.1 Motivation

A large part of the ADVANCE project focus is on formal and simulation-based verification: does the final implementation meet the original specification. There is a requirement, however, to integrate this verification capability 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.

We focus on two primary areas: identifying functional requirements and identifying safety requirements.

2.2 Requirements Analysis

The creation of a consistent system description is a challenging problem of requirements engineering. Formal and informal reasoning can greatly contribute to meet this challenge. However, this demands that formal and informal reasoning and the system description are connected in such a way that the reasoning permits drawing conclusions about the system description.

The primary objective of this part of the work is to integrate Requirements Analysis into the ADVANCE toolset and workflow.

2.3 Safety Analysis

Traditionally, safety analysis is conducted *after the event*. An alternative approach is considered in [Lev12]. Discovering the safety requirements using safety analysis is conducted in parallel with the requirements analysis. In this way, the full requirements of this system are developed much earlier in the development flow.

The primary objective of this part of the work is to develop a method for Safety Analysis which is closely integrated with Requirements Analysis and is integrated into the ADVANCE toolset and workflow.

Chapter 3

Requirements Analysis

3.1 Tracing Requirements into Specifications

The creation of a consistent system description is a challenging problem of requirements engineering. Formal and informal reasoning can greatly contribute to meet this challenge. However, this demands that formal and informal reasoning and the system description are connected in such way that the reasoning permits drawing conclusions about the system description.

In [JHL11], we describe an incremental approach to requirements modelling and validation that incorporates formal and informal reasoning. The elaboration of this work in [Jas12] has been given the name *ProR approach*. Our main contribution is an approach to requirements tracing that delivers the necessary connection that links the reasoning to the system description. Formal refinement is used in order to deal with large and complex system descriptions.

We discuss tool support for our approach of requirements tracing that combines informal requirements modelling with formal modelling and verification while tracing requirements among each other and into the formal model.

A full account of the work described in the report is available in the paper [HJL12].

3.2 A Model for Requirements Tracing

Our approach is based on WRSPM by Gunter et. al. [GJGZ00]. The objective of our approach is to produce a system description of "high quality" by establishing a traceability that allows a systematic validation of the system description and provides robustness with respect to changes in the system

description. It further allows the mixing of formal and informal elements, thereby enabling rigorous reasoning where it is desired.

The approach distinguishes artefacts and phenomena. Phenomena describe the state space and state transitions of an environment and a system, while artefacts describe constraints on the state space and the state transitions.

Artefacts are distinguished into *domain properties* (W), *requirement items* (R), *nonfunctional requirements* (N), *specification elements* (Q), *implementation elements* (P), *design decisions* (U), and *platform properties* (M).

Phenomena are distinguished by whether they are controlled by the system, belonging to set s, or the environment, belonging to set e. Furthermore, phenomena are distinguished by visibility. Environmental phenomena may be visible to the system, belonging to e_v, or hidden from it, belonging to e_h. Similarly, system phenomena belonging to s_v are visible to the environment, while those belonging to s_h are hidden from it. These classes of phenomena are mutually disjoint.

Once a system is modelled following our approach, a number of properties can be verified with regard to the model, one being *adequacy with respect to* Q:

$$\forall e, s \cdot W \wedge Q \Rightarrow R \wedge U \ . \tag{3.1}$$

It says that the specification constrains the world such that the requirements and design decisions are realised. Note that if the world is vacuous, that is, $\neg(\exists e, s_v \cdot W)$, the implication would be trivial to satisfy by any specification. However, if the world W is consistent, then we expect the development method to preserve it: a specification Q must not be permitted to falsify the premise $W \wedge Q$. We expect the specification Q to be feasible assuming W. This can be achieved by construction using refinement, e.g., [Abr10].

The implementation should also satisfy a condition similar to adequacy:

$$\forall e, s \cdot W \wedge M \wedge P \Rightarrow R \wedge U \ . \tag{3.2}$$

If we have already established adequacy, (3.2) can be achieved by refinement:

$$\forall e, s \cdot W \wedge M \wedge P \Rightarrow Q \ . \tag{3.3}$$

The latter formula (3.3) reflects the refinement condition for relations presented in [HJ98]. An additional side condition provides for feasibility. We use the refinement approach of [Abr10] that permits additionally changing the data-representation.

Non-functional requirements depend, in particular, on design decisions. This aspect of non-functional requirements is discussed in [CdPL09]. Design

decisions introduce architectural concepts or constrain the implementation, for example.

$$\forall e, s \cdot W \wedge R \wedge U \wedge Q \Rightarrow N \ . \tag{3.4}$$

We assume that often non-function requirements will not be formal. Hence, formula (3.4) will usually consist of formal and informal artefacts with the conclusion N being informal.

The implications in the formulae (3.1) to (3.4) indicate relationships between specific artefacts. For instance, a specific specification element Q' may imply a specific requirement item R'. We can also say that we can trace requirement R' to specification element Q'. The reference model provides the foundation for our approach of requirement traceability. We also cast the refinement theory of Event-B conceptually into the reference model so that we can trace requirements among formal artefacts, among informal artefacts and across formal and informal artefacts.

3.3 Tracing of Artefacts and Phenomena

In order to trace requirements we need to define relationships between artefacts. Currently, we are not interested in tracing implementation elements P and platform properties M. We focus on the relationship between specification Q and design decision U on one side and requirements items R and N, as well as domain properties W on the other.

We are interested in tracing *justifications* of artefacts, *equivalence* between artefacts, *evolution* of artefacts and tracing of phenomena *used* in artefacts. We discuss the different kinds of tracing in turn.

Tracing Artefact Justification We say B justifies A, or $B \leftarrow A$, if B justifies the *presence* of artefact A. It should be there for a reason. If we read implications like (3.1) from the right to the left we arrive at justifications for the involved artefacts. We say $R \wedge U$ justify $Q \wedge W$. We would like $Q \wedge W$ not to contain more artefacts than necessary in order to establish (3.1). We call a subset SB of the artefacts $Q \wedge W$ such that $SB \Rightarrow R \wedge U$ a *satisfaction base* [KJ10] for $R \wedge U$. We are particularly interested in small satisfaction bases to obtain as precise justifications as possible.

Reading a justification $B \leftarrow A$ in the reverse direction $A \rightarrow B$ we say that A *realises* B.

Tracing Artefact Equivalence In our approach some but not all artefacts may be formal. Often formal artefacts have informal counterparts. If

an artefact A is *formal*, we write A_{F}. We write B_{I} if B is *informal*. When formalising informal requirement items we often get direct correspondences between informal items A_{I} and formal items B_{F}. We say that these items are *equivalent*, denoted by $A_{\text{I}} \leftrightarrow B_{\text{F}}$. If A_{I} and B_{F} are only related by implication following this correspondence, statements about formal world properties may not hold with respect to the corresponding informal world properties. For this to hold we need either $A_{\text{I}} \rightarrow B_{\text{F}}$, that is the formal assumption about the world are not stronger than the the informal assumptions, or equivalence $A_{\text{I}} \leftrightarrow B_{\text{F}}$. Equivalence tells us that the informal domain properties are not stronger than needed for building the system.

Tracing Artefact Evolution A system description *evolves* over time. This may happen due to changing requirement items or due to improvements to the description made by modelling and reasoning. Evolution does not follow logical implication. The best we can do is to record approximately how artefacts have changed over time based on differences between different revisions of the system description. We write $A \rightsquigarrow B$ for A evolves into B. Evolution traces are needed for the benefit of various stakeholders to follow original requirement items into current revision of the system description.

Tracing Used Phenomena The partitioning of the phenomena indicates that it is important to trace phenomena into various artefacts. We need to verify that the various artefacts — informal and formal — only refer to allowed phenomena as outlined in Section 3.2. We have only little means in our hand to achieve consistency between formal and informal artefacts, and this one appears simple and effective. We record references from artefacts to phenomena saying that A uses p, denoted by, $p \in A$. This just means that A makes some statement about p. The management of this relationship can be handled efficiently by proper tool support, as described in Section 3.6.

3.4 Tracing Artefacts into Formal Refinements

If a model is developed by formal refinement, a sequence of machines is obtained, where each new refinement captures some informal artefacts, which can be expressed as invariants. In Event-B refinement, invariants are accumulated along a series of formal refinements. Thus formal refinement permits us to introduce and trace requirements gradually, alleviating a major difficulty when dealing with complex requirements.

Informal Proofs about Formal Models Our approach is not limited to verification by formal proof exclusively. We also permit (and encourage) informal proof. Our aim here is not to formalise everything but to show how formal and informal reasoning can be used together for complex models.

Informal Proofs about Informal Models Artefacts that are not traced into the formal model can not be verified formally. There are various ways to verify them informally. We should certainly not simply ignore them if they do not fit into the current formal scheme. In [HJL12], we provide examples of informal proofs.

3.5 A Process for Requirements Modelling and Validation

In the requirements engineering process we distinguish the different activities of requirements elicitation, requirements specification, system modelling, requirements validation and requirements management [Wie03], as depicted in Figure 3.1.

During the *requirements specification* phase requirements and domain properties are first identified. The objective of *system modelling* is the formal modelling of a subset of the system description as well as the elaboration of the specification elements. The objective of *requirements validation* is to validate the relationship between informal artefacts and formal constructs and to validate the adequacy of the specification elements. We see that *requirements management* as a continuation of modelling and validation in a later phase of a project lifecycle.

In our aproach, we focus on modelling and validation. Common approaches of requirements elicitation could be used to gather requirements in early phases during the process. We do not consider this aspect of the requirements process because it has little influence on modelling and validation.

The phase of modelling and validation consists usually of many iterations between modelling and validation where the collection of all artefacts is validated incrementally. A detailed description of the process can be found in [JHL11].

The Fully Validated System Description When all traces have been validated and claims in the formal model have been verified the system description is considered consistent with respect to

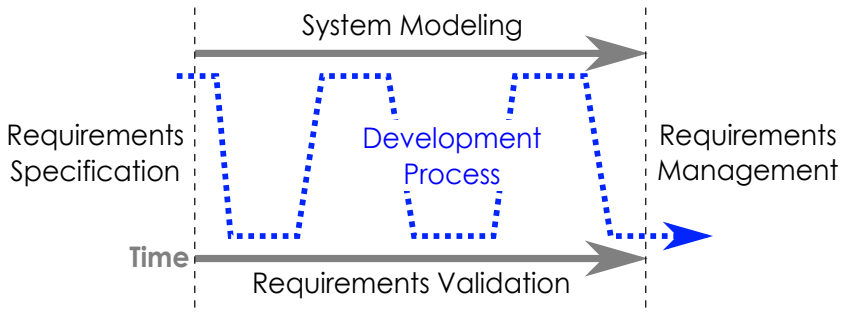

Figure 3.1: The Incremental Development Process

- the allowed references of phenomena by the different artefacts, as described in Section3.2;
- the use of phenomena across different artefacts;
- the traces respecting formulae (3.1) to (3.4);
- the relationship of informal artefacts to formal constructs;
- the verified formal properties of the formal model.

Our approach only classifies artefacts and their relationships. There is no provision for structuring the collection of artefacts as a whole. We acknowledge that the structuring is a relevant issue in practice and rely on approaches complementary to ours to carry this out. For instance, [Kov98] argues that a list of requirements is much easier to understand if they are given a meaningful order. Furthermore, additional structure such as sections or headlines improves readability and scalability of a system description.

3.6 Tool Support

Keeping track manually of large sets of requirements and their relationships is not feasible. For this reason it is mandatory that the method for requirements modelling that we suggest be supported by a software tool. The tool ProR can be extended to achieve this. The generic method-independent characteristics of the tool are discussed in [Jas10].

The genericity of ProR is achieved by means of the Eclipse Requirement Modelling Framework (RMF, http://eclipse.org/rmf) [JG12], which consists of a generic data model for requirements. Despite its genericity, a key objective in the development of ProR has been to support the approach described in this paper.

The generic ProR supports tracing natural language requirements in the form of hierarchical tables. A dedicated column of each table summarises the incoming and outgoing traces.

ProR allows the customisation of its meta-model for concrete notions of requirements. Such a customisation may, for example, consist of adding a new type of requirements with a specified number of typed attributes. The display of the requirements models can also be customised, for example, by showing only selected attributes.

Our approach is supported by means of extending ProR. ProR is integrated with the Rodin formal modelling platform [ABH+10] and ProB [LB08]. The tools Rodin and ProB in combination provide support for proof, model checking and animation of formal models specified in the Event-B notation. The integration with these tools provides support for the central aspect of our approach: to exploit formal reasoning as much as possible for modelling and analysing informal requirements. A screenshot of the resulting tool is shown in Fig. 3.2.

3.7 Conclusion

We have presented an incremental approach for building a system description consisting of formal and informal artefacts. The resulting system description is complemented by a traces between those artefacts that support systematic validation and change management.

The main objective of our approach is the validation of the informal system description. Consequently, the formal model is no end in itself, but only serves as a tool for the rigorous validation of the system description or parts thereof. Specifically, any formal model that we create is located between informal domain properties and informal requirements. The aim of the formal modelling is to ensure consistency of the informal system description. The approach of requirements tracing that we have developed plays a crucial role in this.

The role of the formal model in the overall system development process can vary and depends on a number of factors, including the problem to be solved, the formalism that is chosen, the experience of the team, to name a few.

This work has a strong focus on traceability, and we do not distinguish in principle on whether the artefacts traced are formal or informal. Instead, we focus on classifying the trace as a justification (or its inverse, realisation) and its stronger form, equivalence. This allows us to construct a closed system description that is consistent with respect to the purpose of its artefacts.

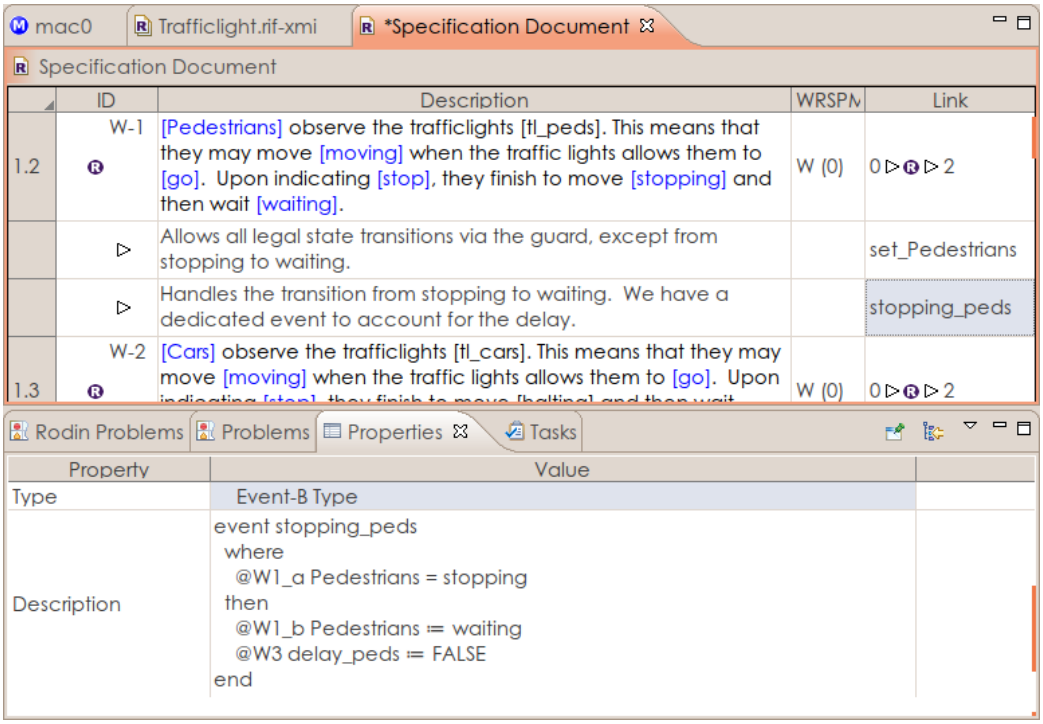

Figure 3.2: The Specification editor of ProR, showing how phenomena are identified in the requirements through color highlighting. The properties view shows the model element that is traced to the selected artifact in the upper pane.

While such an approach cannot identify missing requirements or assumptions, it can ensure that all recorded requirements are realised. Further, the introduction of phenomena allows the systematic creation and maintenance of a glossary, and allows for some rudimentary consistence checks as well.

Finally, the work described here is supported by a tool integrating requirements modelling, ProR, that is integrated with the tools for formal verification by proof, Rodin, and by model-checking an animation, ProB.

Chapter 4

Safety Analysis

4.1 Introduction

To meet the objective of integrating Safety Analysis into the ADVANCE method and toolset, we intend first to adopt the Safety Analysis method proposed by Leveson in [Lev12] and second to use the requirements traceability capabilities of the *ProR* plug-in described in the Requirements Analysis chapter above. We will illustrate the Safety Analysis method with a case study in which the functional and safety requirements of a domestic washing machine will be explored.

4.2 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.

4.2.1 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.

4.2.2 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[YB12]. The phenomena that we shall explore are the

Monitored phenomena, *Commanded* phenomena, *Controlled* Phenomena and *Mode* Phenomena.

4.2.3 Monitored Phenomena

These are the phenomena of the system that will be measured using sensors. The Master Controller will use the values from these sensors to determine its actions.

Drum Water Level

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

Drum Water Temperature

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

Door Position

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

Vibration Level

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

4.2.4 Commanded Phenomena

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

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*.

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

Water temperature setting

The Controller will receive the water temperature setting from the Control Panel: 30, 40 or 60 degrees Celsius.

Start signal

The Controller will receive the start signal from the Control Panel.

4.2.5 Controlled Phenomena

These are the phenomena that are driven by the Master Controller.

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.

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.

Water Control Valves

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

Water Drain Pump

- The Controller activates the water drain pump to meet the water level requirements

Heater

- The Controller activates and de-activates the heater to meet the temperature requirements

4.2.6 Mode Phenomena

The Controller Modes: *Idle, Washing, Rinsing, Spinning*

4.2.7 Discovering the Safety Requirements

The following two quotes from [Lev12] 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.

4.2.8 System-Theoretic Process Analysis

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

- Identify Potentially Hazardous Control Actions
- Derive the Safety Constraints
- 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 [PLH06]. 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.

The Door Sub-system

Consider first a model of the Controlled Door Sub-system as shown in Figure 4.1.

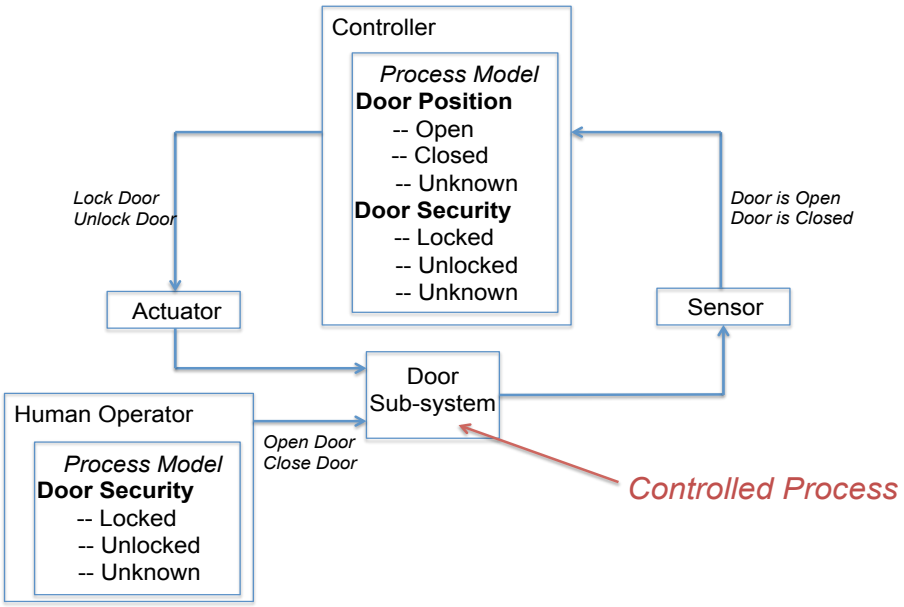

Figure 4.1: The Controlled Door Sub-system

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 I: Identifying Potentially Hazards 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 in Figure 4.2.

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

Figure 4.2: Hazards: Door

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.

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 II: Deriving the Safety Constraints

Three Safety Constraints can be derived from Figure 4.2

- The Door must always be locked when there is water in the Drum
- An *Unlock Door* command must never be issued until the water is fully drained
- 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.

Step III: 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 4.3.

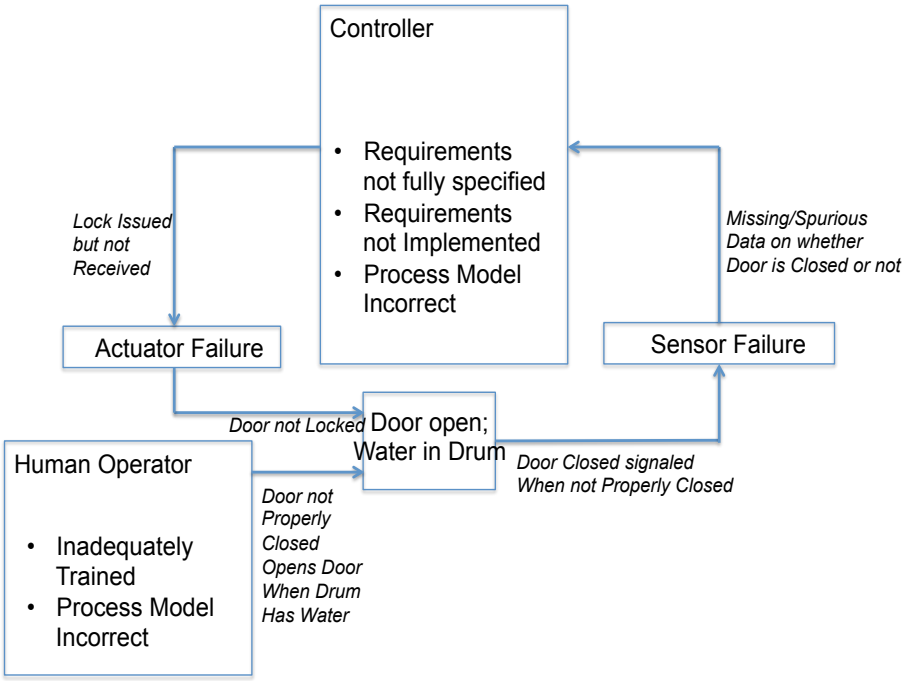

Figure 4.3: Potential causes of Unsafe Actions

The Controller or the Operator can have an inadequate or incorrect process model of the Door Sub-system and actuators and sensors can fail. These potential causes of unsafe actions can be used to both improve the design and inform the test plan.

The Heater Sub-system

We now turn to the Controlled Heater Sub-system as shown in Figure 4.4.

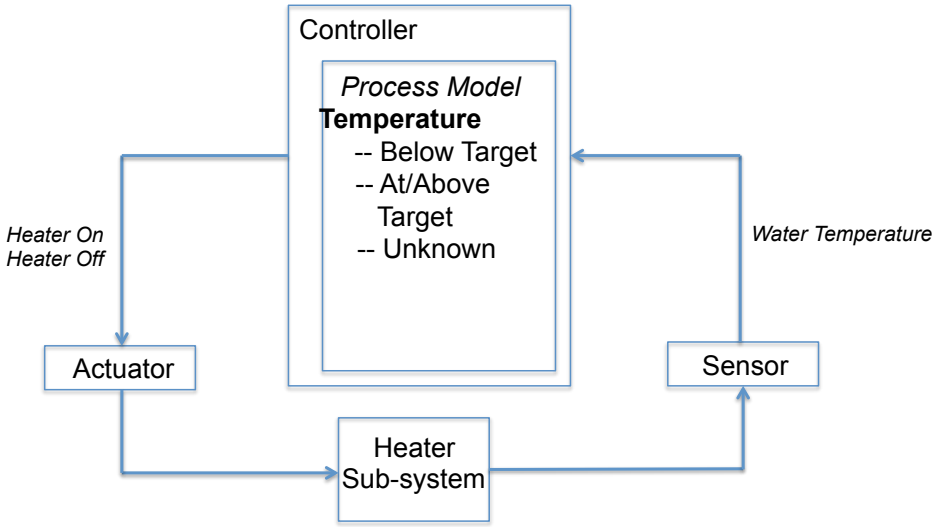

Figure 4.4: The Heater Sub-system

In this case there is a single Controller; the Operator does not control the heater directly. The Controller uses an *Actuator* to switch the heater on and off and a *Sensor* to detect the water temperature.

Step I: Identifying Potentially Hazards Control Actions

For each of the two Controller actions, *Heater Off* and *Heater On*, 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 Figure 4.5.

Controller Action	Not Providing Causes Hazard	Providing Causes Hazard	Wrong Timing or Order Causes Hazard
Heater Off	1. Water already Hot 2. No Water in Drum	Not Hazardous	1. Already Empty 2. Already Overheated
Heater On	Not Hazardous	1. Water already Hot 2. No Water in Drum	Drum has insufficient Water

Figure 4.5: Hazards: Heater

Failing to turn the heater off when the water is already hot or there is no water in the drum is hazardous. Turning the heater off is never hazardous. Failing to turn the heater on is also not hazardous, but turning it on when the water is already hot or there is no water in the drum is.

Turning off the heater too late is hazardous and so is turning it on too early when there is insufficient water in the drum.

Step II: Deriving the Safety Constraints

Several Safety Constraints can be derived from Figure 4.5

- The Heater must always be off if the temperature of the water is higher than a given threshold above the required (selected) temperature. (INVARIANT)
- The Heater must always be off if the water in the drum is empty. (INVARIANT)
- A *Heater On* command must never be issued when the water is already hot. (GUARD)
- A *Heater On* command must never be issued when the water is below the minimum level. (GUARD)

Once again, these natural language guards can be modelled formally with Event-B.

Step III: Determining How Unsafe Control Actions could Occur

Again, we revisit the Controlled Heater Sub-system to determine systematically the potential causes of unsafe actions as shown in Figure 4.6.

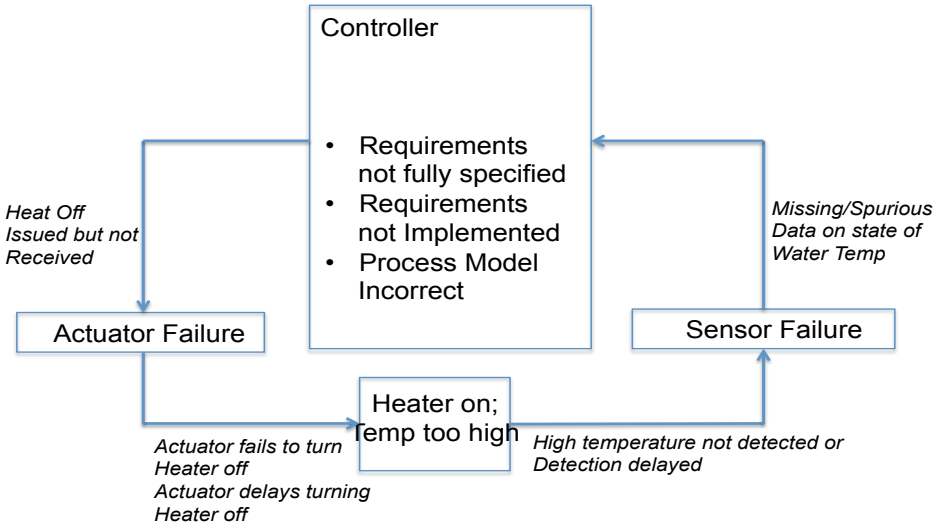

Figure 4.6: Potential causes of Unsafe Actions

The Drum Sub-system

Next, we look at the Controlled Drum Sub-system as shown in Figure 4.7.

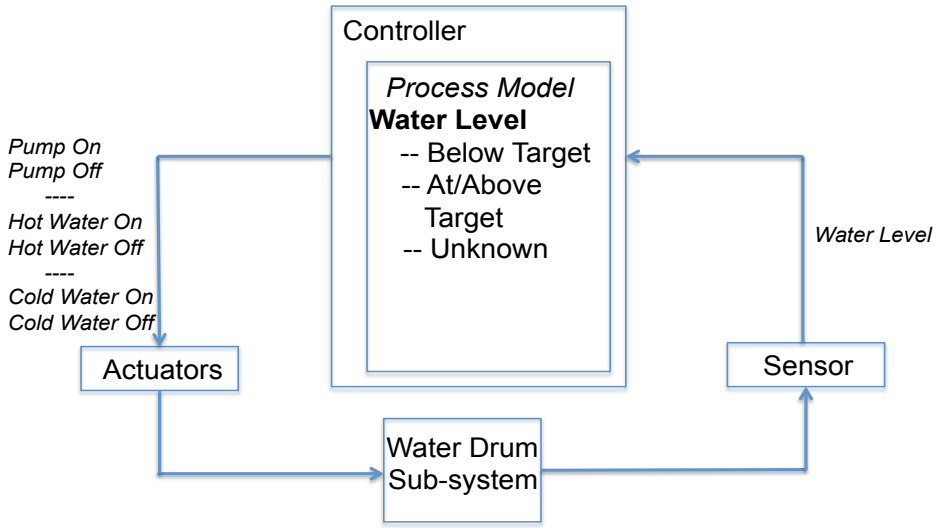

Figure 4.7: The Drum Sub-system

In this case, we need to consider the pump, the hot and the cold water valves.

Step I: Identifying Potentially Hazards Control Actions

First we consider the pump, as shown in Figure 4.8.

Controller Action	Not Providing Causes Hazard	Providing Causes Hazard	Wrong Timing or Order Causes Hazard
Pump Off	Pump operates with no Water in Drum	Water level too high in Drum	Drum Already Empty
Pump On	Water level too high in Drum	Pump operates with no Water in Drum	Drum level already too high

Figure 4.8: Hazards: Pump

and then the water valves, as shown in Figure 4.9.

Controller Action	Not Providing Causes Hazard	Providing Causes Hazard	Wrong Timing or Order Causes Hazard
Hot/Cold Water Off	Water level too high in Drum	Not Hazardous	Drum level already too high
Hot/Cold Water On	Not Hazardous	Water level too high in Drum	Not hazardous

Figure 4.9: Hazards: Valves

Step II: Deriving the Safety Constraints

We then derive the Safety Constraints from Figure 4.8 and Figure 4.9. Here, we just show the *Invariants*.

- The Pump must always be off if there is no Water in the Drum. (INVARIANT)
- The Pump must always be on if the Water level is too high. (INVARIANT)
- The Hot and Cold Water must always be off if the Water level is too high. (INVARIANT)

Step III: Determining How Unsafe Control Actions could Occur

We revisit the Controlled Drum Sub-system to determine systematically the potential causes of unsafe actions as shown in Figure 4.10 and Figure 4.11.

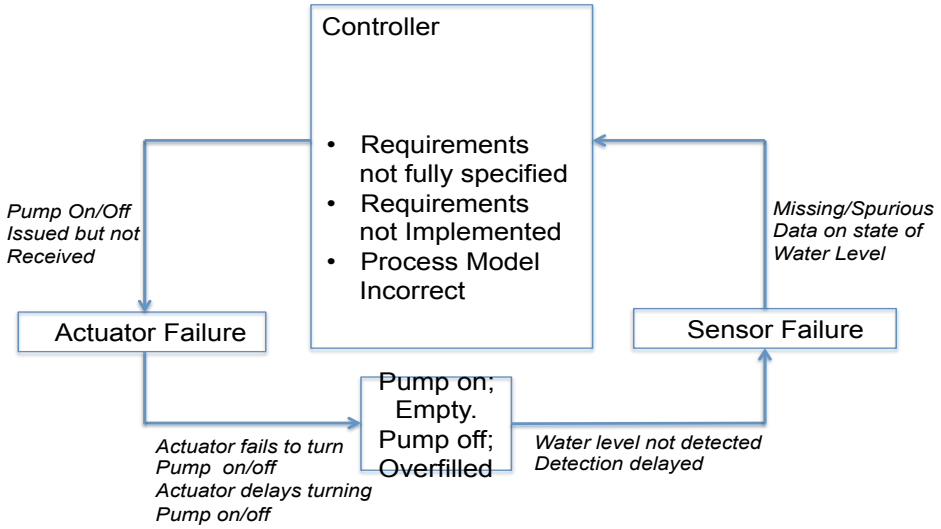

Figure 4.10: Potential causes of Unsafe Actions: Pump

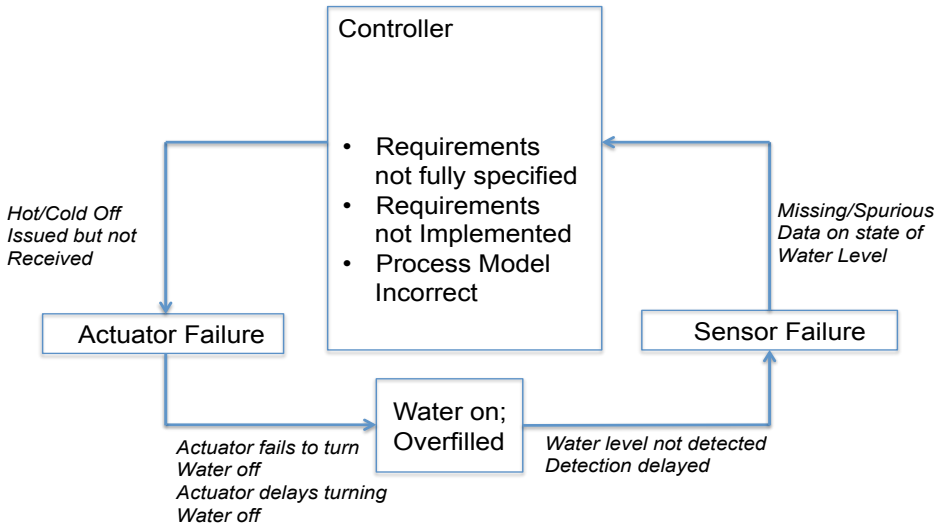

Figure 4.11: Potential causes of Unsafe Actions: Valves

The analysis is, finally, repeated on the Agitator Motor sub-system.

4.3 Summarising the Safety Analysis Method

- The Functional Requirements are developed using the System Phenomena
- The Safety Requirements are derived from the Controlled Phenomena
- The Safety Constraints are then derived systematically from the Safety Requirements, represented in natural language
- The Safety Constraints are represented formally in the Event-B model as invariants and guards

4.4 Tool Support

Since ProR is highly configurable, no special tooling will be needed to support the ADVANCE Safety Analysis method. We will first configure ProR to capture the requirements phenomena and then link the *controlled* phenomena to the safety analysis. We will also link the Event-B model invariants and guards to the safety requirements. In this way we will leverage the traceability facilites already provided by ProR.

Chapter 5

Summary

We have shown that we have developed a method in ADVANCE for capturing System Requirements and providing traceability between those requirements and a formal specification of the system in Event-B. We provide tool support with ProR which has been shown to be both flexible and extensible. We have also developed a method for capturing Safety Requirements which is integrated with Functional Requirement capture and will also use the ProR facility. In future work, we plan to configure ProR to support this integrated requirements capture method, using the Washing Machine Case Study to validate our approach.

Bibliography

[ABH+10] J.-R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

[Abr10] J.-R. Abrial. *Modeling in Event-B – System and Software Engineering*. Cambridge University Press, 2010.

[CdPL09] L. Chung and J. C. Sampaio do Prado Leite. On non-functional requirements in software engineering. In A. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. K. Yu, editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *LNCS*, pages 363–379. Springer, 2009.

[GJGZ00] C. A. Gunter, M. Jackson, E. L. Gunter, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17:37–43, 2000.

[HJ98] C. A. R. Hoare and H. Jifeng. *Unifying Theories of Programming*. Prentice Hall, 1998.

[HJL12] S. Hallerstede, M. Jastram, and L. Ladenberger. A method and tool for tracing requirements into specifications. Submitted to Science of Computer Programming, 2012.

[Jas10] M. Jastram. ProR, an open source platform for requirements engineering based on RIF. *SEISCONF*, 2010.

[Jas12] M. Jastram. *The ProR Approach: Traceability of Requirements and System Descriptions*. Inaugural-Dissertation. CreateSpace, 2012.

[JG12] M. Jastram and A. Graf. ReqIF – the new requirements standard and its open source implementation Eclipse RMF. Technical report, Commercial Vehicle Technology Symposium, 2012.

[JHL11] M. Jastram, S. Hallerstede, and L. Ladenberger. Mixing formal and informal model elements for tracing requirements. In *AVOCS 2011*, AVOCS 2011, 2011.

[KJ10] E. Kang and D. Jackson. Dependability arguments with trusted bases. In *RE*, pages 262–271. IEEE Computer Society, 2010.

[Kov98] B. L. Kovitz. *Practical software requirements: a manual of content and style*. Manning, 1998.

[LB08] M. Leuschel and M. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.

[Lev12] N.G. Leveson. *Engineering a safer world: Systems thinking applied to safety*. MIT Press (MA), 2012.

[PLH06] S.J. Pereira, G. Lee, and J. Howard. A system-theoretic hazard analysis methodology for a non-advocate safety assessment of the ballistic missile defense system. Technical report, DTIC Document, 2006.

[Wie03] K. E. Wiegers. *Software Requirements: Practical Techniques for Gathering and Managing Requirements throughout the Product Development Cycle*. Microsoft Press, 2nd edition, 2003.

[YB12] S. Yeganefard and M. Butler. Control systems: Phenomena and structuring functional requirement documents. 2012.