Technical Report No. 2008/01

Problem Oriented Formal Requirements
Modelling & Analysis

Derek Mannering
Jon G. Hall
Lucia Rapanotti

4th March 2008

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk
Problem Oriented Formal Requirements
Modelling & Analysis

Derek Mannering¹, Jon G. Hall², and Lucia Rapanotti²

¹ General Dynamics UK Limited
² Centre for Research in Computing, The Open University

Abstract. Safety is a factor of a system operating in a particular environment. Therefore the requirements engineering method used must adequately distinguish between the desired behaviour of the requirements and the extant properties of the environment that are relied upon. This paper will show that Problem Oriented Software Engineering (POSE) has these capabilities, and can be used in conjunction with the Alloy formal method to achieve a number of related goals. That is, POSE and Alloy allow the formalisation of the requirements transformation process to assist in (a) the derivation of implementable requirements, (b) the production of a formal requirements specification model and (c) the task of performing a preliminary hazard analysis.

1 Introduction

Formal specification is required by a number of safety [1] and security [2] standards for their highest integrity applications. The use of such techniques is not universal, but to some they provide a “gold standard” [3] of what can be achieved. However researchers have noted that the perceived benefits of applying formal specification techniques are not being realised [4], [5]. This has been attributed to a number of causes including (a) a lack of methodological guidance for their application [4]; (b) that formal techniques are not well integrated with the system analysis phase, there remains the need to analyse system requirements and its environment before producing the specification [4]; (c) that formality has difficulty in distinguishing wished for system properties from extant environment properties [5]; and (d) that formal specification languages are not well suited to the requirements engineering (RE) tasks as they only address the “what” question [6].

As an approach, many have advocated extending the scope of formal techniques through the development of an integrated RE process based on formality covering elicitation, modelling and analysis, communication and validation, and also stresses the importance of the environment [6][4]. This has been successful in extending the reach of formal methods, but has not overcome the disconnect of Turski’s problem [7]: that real-world domains are not necessarily expressible in any single linguistic system; and that the notion of mathematical (logical)
proof does not apply to them. We believe that this disconnect manifests itself in
the following ways:

– formal methods neglect of the validity of (safety) requirements, i.e., (within
the formal method) they are typically not derived through environmental
analysis and are not traceable with respect to the system as designed and
implemented;

– make questionable well-foundedness of assumptions about the system, its
operating environment or modes of use;

– prevent reflection—often and early—on the problem being solved and vali-
dation of progress towards solution.

In summary, because of the Turski disconnect, from within a formal method
we can never be quite certain whether the real problem is being solved. Formal
methods is the best way we have of building zero defect software (for instance,
[8]); but effort and investment are high for this high pay-off. And, if formal
methods are to have a role in the new risk-driven approaches advocated by
modern standards, it will not be sufficient that formal methods can deliver zero-
defect software—the secondary risk—that we will produce zero-defect software
for the wrong problem—needs to be accepted and managed.

In this paper we provide a structured approach around a formal method
by which the secondary risk can be managed. We show how Problem Oriented
Software Engineering (POSE) [9] in conjunction with the Alloy formal method
[10] provides structures that (a) support the production of formal requirements
models and their analysis as part of an integrated RE process and (b) allow that
formal analysis to feed backwards and forwards into the non-formal domain.
Careful adoption of our techniques may lead to the tighter cross-coupling be-
tween non-formal and formal developmental components needed to be as sure as
possible that the right problem will be solved. Additionally, it brings the benefits
of validation to formal methods and the benefits of verification to non-formal
methods.

The example we pursue was formally analysed in [3]. A number of the interest-
ing themes introduced by that paper are developed and expanded in association
with other related work; In particular the work on the formal safety analysis of
formal models reported in [11], [12], [13], [14]. The paper is organised as follows:
brief descriptions of the background to the paper are given in Section 2. Section 3
applies the POSE and Alloy combination on the Sluice Gate case study. Section
4 includes related work. Section 5 contains a discussion and conclusions.

2 Background and Overview

POSE is the software engineering ‘instance’ of Problem Oriented Engineering
(POE) [9]; a natural framework for engineering design. In POE, engineering de-
sign is seen as a form of problem solving; POE allows problems to be transformed
into problems that are easier to solve, or that will lead to other problems that
are easier to solve. Each transformation requires a justification: simplifying only
slightly, the justification shows how (and why) a solution to the transformed
problems is related to a solution to the original problem. The combination of
the justifications is an argument that the solution is adequate as a solution to
the original problem.

When software assurance and developmental risk management is paramount,
the POE process pattern [15] provides guidance to the developer for the inter-
leaving of problem and solution exploration and validation steps. In addition, it
has the following characteristics: (a) it provides a vehicle for the assurance case
driven design, with documentation and analysis of the rationale for decisions
(for the importance of integration with assurance cases see [16]); (b) it allows
for the explicit consideration of the risks involved in design; (c) it allows rich
traceability between requirements, domain assumptions and system components.

[17] describes a defining instance of the POE process pattern for safety-critical
development (refer to Figure 1). The activities there include the following: (a)
problem exploration activities by which increasing knowledge and detail of the
environment and requirements of the solution are captured, prior to problem
validation (activity 2); (b) solution exploration steps by which an architecture
(logical and/or physical) for the solution is chosen, and used to transform the
problem; (c) preparation for solution validation through a Preliminary Safety
Analysis (PSA), a form of safety analysis conducted to ensure a feasible solution
structure has been chosen.

Successfully negotiating problem validation choice point (labelled 2) trans-
fers the risk of the wrong problem being solved onto the validator—typically the
customer or similar—after problem exploration (labelled 1) has completed. The
choice point (labelled 4) is informed by the PSA to determine whether (a) the
current architecture is viable as the basis of a solution; or (b) whether backtracking
and (re-)development of the problem (activity 1) and/or another candidate
architecture (activity 3) should be chosen.

The "Four Dark Corners" paper [18] provides much of the foundation work
for the problem oriented approaches such as Problem Frames (PF) [7] and POSE.
The overall system requirements ($R$) are expressed in terms of the environment
before and after the machine application to be designed (called "machine" hence-
forth) is applied to it. The aim of this is to avoid implementation bias that can
occur in some forms of model-based specification. In this approach the overall
requirements are expressed in terms of the relevant phenomena of the machine
environment and this can be far removed from the phenomena accessible to the
machine. This means there is a gap between the overall requirements expressed
in terms of environment phenomena and the requirements specification relevant
to the machine. The goal of the design task is to "uncover" this machine spec-
ification by making the overall requirements implementable - i.e. expressed in
terms of phenomena applicable to the machine. [18] makes the important dis-
tinction between optative properties that relate to the machine requirements,
and indicative properties that relate to properties provided by the environment.
In this terminology the design task is to extract the optative machine speci-
fication that in conjunction with the indicative properties of the environment
Problem exploration: Context and Requirements Interpretation

Solution exploration and PSA

Not PSA ok

Not valid

valid

Fig. 1. POSE Safety Pattern, as an instance of the POE Process Pattern [15]

satisfies the overall system requirement. This can be represented by the sequent, $W, S \vdash R$. In this sequent $W$ represents the indicative domain properties of the environment, $R$ represents the overall system requirement and $S$ is the optative machine requirements specification that in conjunction with $W$ satisfies $R$.

Making the overall requirements implementable is addressed in [18] by identifying three types of non-implementable requirements. These types of requirement are analysed in detail in [19], which shows how the POSE problem progression transformation (PPT) can be used in the PSA step to provide the mechanism by which domain properties are used in the formation of implementable requirements; i.e. the transformed requirements are defined in terms of machine phenomena.

This work traces through the steps in [3] to show that the POSE safety pattern and Alloy together provide the process structure that supports the production of a valid solution. It also develops the domain fault ideas contained in [3] and considers what is needed for an adequate composition (section 3.6). In addition it considers the important concepts of (a) specific concerns (section 3.2), and (b) applying formality to safety analysis (section 3.5). Using a formal model to provide enhanced, formal safety analysis capabilities has emerged as an important research topic in recent years. First the system formal model is developed and validated to establish nominal correctness, then failure injection is applied to investigate the system’s safety capabilities. One approach identified a manual method for injecting failures [11], but other work noted the increased workload problem on large systems and developed tool support for automatic injection [12], [13]. A potential problem is that if the model is inadequate then
so will be the analysis, therefore [14] encourages independence and diversity in
the application of safety analysis techniques to ensure adequate coverage.

3 Sluice Gate Case Study

The case study is based on the Sluice Gate problem described in [20] and used
as the example in [3]. The structure of the development will follow that of [3] to
allow a direct comparison to be made.

3.1 Sluice Gate Problem Initial Description

The first step in the POSE safety pattern is Context & Requirement Interpre-
tation (refer to Figure 1) which involves understanding the requirements for
the system and the properties of the environment the system operates in (most
likely with the assistance of the problem validator). The aim is not to rush into
producing a specification, but rather to understand and investigate the problem
context first ensuring that the correct problem has been identified. This stage
will also address the wider system requirement scope issues identified in [3]. We
will assume, for the sake of argument, that problem exploration has produced
the following description of the Sluice Gate behaviour and its environment in

collaboration with the customer\(^1\):

The Sluice Gate controls the flow of irrigation water in a farming envi-
rironment. The customer’s goal is that the Sluice Gate should be open for
a set time each hour to allow irrigation water to flow, at other times it
should be closed. The open and closed times are designed to be variable
(within defined constraints) to allow the customer to set the system up
so as to provide an optimal irrigation scheme. This goal is termed the
customer irrigation policy.

The natural language requirement for the Sluice Gate problem, RSG, can be
expressed as follows.

1. In any defined repetitive time period the gate should be open for \(D_{open}\) minutes
   and closed for \(D_{clos}\) minutes. Note that \(D_{open} + D_{clos} < \text{period}\) to allow for the transit
   of the gate up or down.

2. The gate is opened or shut by applying appropriate motor control commands
to the GA. The commands are \(mgoup\) to open the gate, \(mdown\) to close the gate and
   \(mstop\) to stop the motor moving.\(^7\)

The design task is to produce a controller that implements the desired cus-
tomer irrigation policy.

In parallel with the validatable problem description, a justified consistent initial
system concept will need to be defined that can feed into any subsequent formal-
isation. That that emerges from this work is shown in Figure 2. It consists of a

\(^1\) Note the use of natural language, not formality, because our intention is to present
the problem validation to the the customer for validation.
controller (CL) that interfaces to the gate assembly (GA) with the motor control, CL!Mcon, phenomena and gate status, GA!Stat phenomena. The CL! means that Mcon is controlled by the CL domain and so on. The Alloy model can be directly constructed from the POSE description using the process introduced in [21]. This involves partitioning the model into the following phases: (a) phenomena data type definition, (b) domain State data type definition, (c) define domain behaviour predicates, and (d) set up Simulation time trace, simulation and proof check structures.

The modelling work has shown that each of these phases follows a similar pattern form and part of a basic simple model for $P_{Initial}$ is shown in Figure 3. The Mcon phenomena controls the direction of the motor either group to move the gate up, down to move the gate down or stop to stop the motor (and hence gate) from further movement. Note that the system is initialised to a known safe state with the motor stopped (stop) and the gate closed (bot) as shown by pred init[] in Figure 3. Note that the full Alloy listings are too voluminous for this paper, so are available as a Technical note from the OU website.

```alloy
//### Phenomena Data Type ###
abstract sig Status {}
// Define Gate Status Component
sig Stat extends Status
one sig top, bot, trans extends Stat
abstract sig Motor {}
// Define Motor control Component
sig Mcon extends Motor{}
one sig goup, down, stop extends Mcon

//### Domain State Data Type ###
// Define the single Controller, CL State - Motor control
one sig CL { mctrl : Mcon -> Time}

//### Domain Behaviour Predicates ###
// CLBehave models the behaviour of the GA controller.
pred CLBehave(t,t' : Time) { controlrequire[t, t'] }

//### Define Simulation Time Trace Structure Model ###
pred init[t : Time] { GA.gstat.t = bot and CL.mctrl.t = stop }
fact traces { init[Ti/first[]] all t : Time - Ti/last[] | let t' = Ti/next[t] | CLBehave[t,t'] and GABehave[t,t'] }
```

The Stat phenomena provides the status of the Sluice Gate - whether it is fully at the top of its travel, top (gate open), fully at the bottom of its travel bot (gate closed) or in transit from one to the other trans. The domain state data types for CL is shown on Figure 3, GA is not shown but has a similar form. These record the output phenomena of the domain as a function of time. The corre-
sponding domain behaviour predicate for CL, \(CL\)\textit{Behave} is shown on Figure 3, the behaviour for \(GA\) is not shown, but it has a similar structure. The customer’s required irrigation policy was encoded as the predicate \(controlrequire[t, t']\). This policy was also encoded as an assert predicate – not shown on Figure 3, but similar to \textit{ensurebreakok} discussed in section 3.2. A proof was then successfully conducted to show that the model behaviour (primarily \(controlrequire[t, t']\)) satisfied this policy assert predicate. The timing trace model used the predicate \(init[]\) and fact \textit{traces} shown at the bottom of Figure 3, and is based on the standard form presented in Chapter 6 of [10]. In this simple model time is modelled in discrete steps, such that if the motor is commanded to \textit{goup} at time \(t\) then it will have moved the gate to the top by the next time instant \(t+1\). Similarly for \textit{down}, hence \textit{trans} was not used in the model. A more complex model was developed that modelled the gate whilst it was in transit from top to bot or vice versa, in this model \textit{trans} was used. This more complex model was shown to have the desired behaviour, albeit at much longer execution times!

At this point the customer requirement, \(RSG\), controls the gate \textit{open} or \textit{close} in line with the irrigation policy by controlling the motor using \textit{mgoup}, \textit{mdown} and \textit{mstop}. The controller \(CL\) uses the sensor readings (\(top\), \(bot\), \(trans\)) to determine the position of the gate and controls the motor using \textit{goup}, \textit{down} and \textit{stop}. Therefore \(RSG\) and \(CL\) are defined in terms of different phenomena, and the task is to transform \(RSG\) so that the machine specification can be extracted.

3.2 Improving the Model: The Breakage Concern

So far the development is rather idealised and as noted by Jackson [22] each problem raises a number of specific concerns that must be addressed. These include consideration of the (a) initialisation issues, (b) breakage concern, (c) reliability concern and (d) information deficit concern. The initialisation issue was mitigated by starting the system in a known safe state, as detailed above. To allow comparison with [3], we will now just concentrate on the breakage concern, but noting that the other concerns are also likely to be relevant in many developments. The motor could be damaged by switching directions without an intervening pause. To avoid this, the motor must be stopped for a period of not less than \textit{motor\_shutdown} when a change of motor direction is required. A second restriction concerns the situations where the motor has driven the gate to the top or bottom of its travel, but motor drive is still being applied. If this is allowed to continue then the motor could burn-out. To avoid this motor drive power must be removed within a time period of \textit{motor\_limit} after the gate has reached the top or bottom of its travel. These breakage concerns must be incorporated into the simple model of Figure 3. To implement \textit{motor\_shutdown} and \textit{motor\_limit} requires more knowledge of the past behaviour of the motor control, this results in extra state, \(prec\), being added and the behaviour modified as follows:

```latex
one sig CL { mctrl : Mcon -> Time, prec : Mcon -> Time} {}

pred CLBehave(t,t':Time) {breakok[t,t,t']} and CL.prec.t'=CL.mctrl.t }
```
The `breakok[]` predicate encodes the complex behaviour required to satisfy the breakage concerns, it has the form shown below:

```alloy
pred breakok [last, now, next : Time] {
    (CL.prec.last = stop and CL.mctrl.now = stop => (GA.gstat.now = top => CL.mctrl.next = down or CL.mctrl.next = stop)) and
    (CL.prec.last = stop and CL.mctrl.now = stop => CL.mctrl.next = stop) and
    (CL.prec.last = stop and CL.mctrl.now = goup => CL.mctrl.next = stop) and
    (CL.prec.last = stop and CL.mctrl.now = down => CL.mctrl.next = stop) and
    (CL.prec.last = goup and CL.mctrl.now = top => CL.mctrl.next = down or CL.mctrl.next = stop)) and
    (CL.prec.last = goup and CL.mctrl.now = down => CL.mctrl.next = stop) and
    (CL.prec.last = goup and CL.mctrl.now = goup => CL.mctrl.next = stop) and
    (CL.prec.last = goup and CL.mctrl.now = down => CL.mctrl.next = stop) and
    (CL.prec.last = goup and CL.mctrl.now = down => CL.mctrl.next = stop) and
    (CL.prec.last = down and CL.mctrl.now = stop => CL.mctrl.next = stop) and
    (CL.prec.last = down and CL.mctrl.now = goup => CL.mctrl.next = stop) and
    (CL.prec.last = down and CL.mctrl.now = down => CL.mctrl.next = stop)
}
```

Basically, two successive motor stop commands are required before the motor is allowed to be commanded up or down, and then it can only be commanded to move the gate away from its current position. Thus if at time t the gate has just been commanded to fully open, `GA.gstat.now = top`, then the motor is only allowed to be controlled down after two successive periods of stop - this is captured in the first couple of lines of the predicate, the rest of the predicate commands `stop`. Other more general forms of `breakok[]` can be defined, but the one given captures the essence of the required behaviour. Simulation of the revised Alloy model showed that it had the desired behaviour. The breakage concerns were then encoded as asserts and validated against the model. For example, `ensurebreakok` was asserted in the model and found to be consistent; `ensurebreakok` has the form:

```alloy
assert ensurebreakok { all t1 : Time - Ti/last[] - prev[Ti/last[]] | ( some t2 : Time, motor_shut : Int | # (nexts[t1] - nexts[t2]) = motor_shut and 1t(t1, t2) and motor_shut = 2 and
    (CL.mctrl.t1 = goup => (#(down<(CL.mctrl.>(nexts[t1] - nexts[t2]))) = 0)) and
    (CL.mctrl.t1 = down => (#(goup<(CL.mctrl.>(nexts[t1] - nexts[t2]))) = 0)) )
}
```

In this logic “a => b” means “If a Then b”, the # operator finds the cardinality of the set, `<:` is domain restriction, `:` range restriction and `nexts[t1]` means the set of all the elements that follow t1 in the ordering. Therefore the fragment

```
# (down<br:(CL.mctrl.>(nexts[t1] - nexts[t2])) = 0)
```

finds the trace of all the CL.mctrl control values in the period after t1 up to t2 set of times using range restriction. This trace is then restricted to the domain containing only the `down` element, and the number of elements in this set is zero. Overall, this predicate states that for all times t1 where a motor control of `goup` is issued then there will be no `down` issued in the period of `motor_shut` after t1. Similarly, when t1 corresponds to a `down` command there will be no `goup` commands for a period of `motor_shut`. Therefore this predicate implements the `motor_shutdown` breakage concern. Incorporating the breakage concerns has resulted in a more complex, but more realistic Sluice Gate behaviour model.

### 3.3 Solution Exploration
The next task is Solution Exploration (part of Activity 3 in Figure 1). This involves identifying an architecture that is capable of satisfying the needs of the Sluice Gate problem, changing the problem from $P_{Initial}$ into $P_{Arch}$ in the process. The main issues in this problem transformation concern (a) driving the motor and (b) reading the gate position sensor. Both of these are signal conditioning problems that have been encountered before on similar systems and can be solved by introducing suitable interface units as shown in Figure 4. The sensor interface domain (SI) reads the status of the gate position from the GA domain and passes this information to the controller system (CS), which is the system to be designed. The motor interface domain (MI) takes the control logic from the CS and conditions it to provide the power to drive the motor correctly. The SI and MI are given domains in that they are standard units requiring only standard configuration modifications to be useful in the Sluice Gate architecture - i.e. no extensive design work is required on them. In contrast, the control logic of the CS has to be designed.

The justification for introducing this architecture and transforming the problem to $P_{Arch}$ is “the architecture is known to have been successful on similar system designs”. Further, the introduction of the architecture has not resulted in any change to the requirements, so that $RSG = RSG'$. In a similar process to that used for $P_{Initial}$, the POSE description for $P_{Arch}$ can be directly translated into its corresponding Alloy model. The SI and MI domains have known, standard behaviour descriptions and these were used directly in the model. Simulation and proof work demonstrated that it had the required behaviour.

3.4 Problem Progression

The next step in the POSE safety pattern is the PSA which has two components (a) problem progression and (b) the safety analysis (Figure 1). Problem progression involves applying the PPT to remove domains and to transform the requirements. In this example only one domain, GA, needs to be removed before the transformed requirements apply directly to the to be designed control machine, CS. The result of applying the PPT is to transform the problem from $P_{Arch}$ into problem $P_{Prog}$ as shown in Figure 5. The GA domain has been removed and the requirements transformed from $RSG$ ($RSG'$) into $RSG$.

The requirements are transformed as detailed in [19] such that references to the removed domain (GA) are re-written in terms of the remaining domains:
1. In any defined repetitive time period the sensor should be top for $D_{open}$ minutes and bot for $D_{close}$ minutes. Note that $D_{open} + D_{close} < \text{period}$ to allow for the transit of the gate up or down.

2. The sensor is set to top or bot applying appropriate motor control commands to the MI. The commands are $\text{goup}$ to set the sensor to top, $\text{down}$ to set the sensor to bot, and $\text{stop}$ to stop the motor moving.

The justification for the transformation is that (a) the sensor readings top/bot correspond directly to the gate being open/shut respectively and (b) the control commands $\text{goup}$, $\text{down}$ and $\text{stop}$ will control the motor as the commands $\text{mgoup}$, $\text{mdown}$ and $\text{mstop}$. This justification captures the behaviour required of the removed domain (GA) as an assumption that must be validated as part of the development. At this point the requirements $\text{RSG}''$ refers to the same phenomena as $\text{CS}$, hence the machine specification, $S$, for the $\text{CS}$ can be derived from: $W, S \vdash \text{RSG}''$, where $W$ represents the known behaviour of the SI and MI domains. This knowledge about $S$ was used in the derivation of an Alloy model, which used the same method as described in section 3.1. The model was simulated and proofs carried out to show that it had the desired behaviour.

3.5 Safety Analysis

The safety analysis in [3] was informal and mixed sub-system hazards with failure modes of the system that can cause these hazards. This work follows a more structured analysis along the lines of ARP4761 [23] which envisages a hierarchy of safety analysis tasks consisting System Safety analysis, Functional Hazard Analysis (FHA) and Preliminary System Safety Analysis (PSSA) - corresponds to the PSA in the POSE Safety pattern (Figure 1). The FHA identifies the hazards that apply to the system under analysis, the role of the PSA (PSSA) is to investigate to what extent this system contributes to these hazards and to identify additional hazards if they exist. In this analysis it was decided to use the formal Alloy model in conjunction with modified HAZOPS guide words [24] applied to all the phenomena to achieve a more thorough investigation. The rationale being that the phenomena determine what impact one domain can have on another and the guide words provide a complete semantic set of what can happen to the phenomena being investigated. Therefore the safety analysis is using the same model as the development. This is efficient because there is no need to develop a separate safety model that has to be validated, and the risk of introducing anomalies into the safety model is eliminated.
The higher level safety analysis identified critical hazards – No Irrigation achieved & Land flooded – and significant hazards – Customer irrigation policy not satisfied & damage to Sluice Gate system components. The system was then analysed using the guide words: No, More, Less, AS Well As, Part Of, Other Than, Reverse, Early, and Late. These were applied to the phenomena in the model and Table 1 shows the subset of the results obtained containing critical or significant hazards.

<table>
<thead>
<tr>
<th>Guide</th>
<th>Phenomena</th>
<th>Deviation</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>No goup</td>
<td></td>
<td>Motor will not raise gate</td>
<td>No irrigation</td>
</tr>
<tr>
<td>No down</td>
<td></td>
<td>Motor will not lower gate</td>
<td>Flood</td>
</tr>
<tr>
<td>No top</td>
<td></td>
<td>System does not know gate fully open</td>
<td>Breakage concern, motor damage</td>
</tr>
<tr>
<td>More goup</td>
<td></td>
<td>Motor drive exceeds specification</td>
<td>Possible motor damage</td>
</tr>
<tr>
<td>Less down</td>
<td></td>
<td>Motor drive less than specification</td>
<td>Slow/no operation, policy not met</td>
</tr>
<tr>
<td>As Well As goup</td>
<td></td>
<td>Intermittent switching goup/down.</td>
<td>Possible motor damage</td>
</tr>
<tr>
<td>Part Of down</td>
<td></td>
<td>Little power to drive the motor</td>
<td>Slow/no operation, policy not met</td>
</tr>
<tr>
<td>Other Than goup</td>
<td></td>
<td>down instead of goup</td>
<td>Breakage concern, motor damage</td>
</tr>
<tr>
<td>Early goup</td>
<td></td>
<td>gate raised earlier than expected</td>
<td>More water, policy not met</td>
</tr>
</tbody>
</table>

Table 1. Important HAZOPS Results

The next step in the safety analysis process was to investigate manually injecting failures into the formal Alloy model using techniques similar to those in [11]. This was performed as a two stage process, the first followed an FMEA-like form [25] the second followed FTA [26] principles. The failure injection was applied to the interface phenomena which are directly identified in the modelling approach used (see Figure 3) by changing the assignment. For example, the correct setting of \( G_A.gstat.t' = \text{top} \) was mutated into failure forms such as \( G_A.gstat.t' = \text{bot} \) and so on. The simulation was re-run for each failure mode, and then for multiple failure modes. This was found to be a time consuming process and for this reason systematically investigating intermittent failure (by switching failures on and off at a variety of set times) was not performed in depth. As noted by [12], this process needs to be automated, and given the structure of the model this would be a straightforward process. The results obtained were comparable with those obtained from the HAZOPS work.

The FTA-like analysis was based around using the proof checks to show that asserted properties were or were not satisfied by the model. The process followed consisted of (a) encoding the hazard to be checked as an assert predicate (e.g. \( \text{ensurebreak} \) in section 3.2), (b) running the nominal model to show that it is not satisfied, (c) injecting failures to produce a mutated model as described above and (d) re-running the analysis to detect those cases where the mutated model is satisfied. These latter cases (effectively cutsets) identify those failures that can cause the hazard. For example, the critical hazard “no irrigation” will occur if the gate is never opened. This can be encoded as \( \forall t_1 : \text{Time-Ti/last}[\] | \( G_A.gstat.t_1 = \text{bot} \) and failures such as \( G_A.gstat.t \) never being \( \text{top} \) would cause this hazard to be satisfied. This analysis was less time consuming than the others,
but would still benefit from automation. The results obtained were comparable with those obtained from the HAZOPS and FMEA work.

3.6 Composition Issues

So far we have designed a system that implements the customer’s irrigation policy and takes care of the Breakage concern – the IP&BC system. The safety analysis identified a number of hazardous failure modes that indicate a health monitoring system (HMS) needs to be “added” to the system. This follows the sequence from [3]. There is insufficient space to run through the detail of how the HMS is designed as a separate POSE entity, but it is important to consider how it is composed with the IP&BC. The role of the HMS is to monitor the sensor and motor behaviour and to raise an alarm if it is detected to be anomalous as indicated by the safety analysis results of Table 1. An example of anomalous behaviour would be where the sensor indicates top a time of margin after the motor has been commanded down; where a typical motor would be expected to move the gate up or down well within margin time units. This corresponds to the “Motor drive less than specification” deviation in Table 1, and could be caused by a stuck sensor or a failed motor. In either case, the HMS should switch the motor off and raise an alarm.

The question arises as to how should the IP&BC and the HMS be composed? A number of possible composition patterns exist [20], but it is clear from the intended operation that the HMS should override the IP&BC if the former detects anomalous behaviour. This results in the following derived requirement to control the composition:

“The IP&BC shall control the GA unless the HMS detects anomalous behaviour as defined by the safety analysis (Table 1), at which point the HMS shall turn off the motor and raise an alarm”

4 Related Work

The POSE notion of problem fits well with the Parnas 4-Variable model [27] which is well suited to defining embedded critical applications. The 4-Variable model also forms the basis of the SCR [28] and SpecTRM [29] methods, both of which have toolsets for the development of safety systems. Recent work indicates that POSE will interface to these methods.

KAOS [6] [30] has a goal-directed formalised process for requirements elicitation that supports formal modelling. It has strategies for deriving implementable requirements, de-idealising over optimistic goals and managing goal conflict. However, although it does support the distinction between the extant properties of the environment and the desired behaviour of the requirements, this is not as strongly represented as with the problem oriented approaches. The influence of the model-based safety analysis work ([11], [12], [13], [14]) has already been discussed in section 3.5. The agenda-based methods [4] provide guidance to cover
all the important steps of elicitation, consistency, correctness, due consideration of the environment, dealing with conflict and validation. However, they do not benefit from the transformations and structuring provided by POSE which facilitates the model building process (as shown in the case study).

5 Discussion and Conclusions

Previous research has indicated that (a) formal modelling should be an integrated part of the RE process, (b) the formal specification should be derived from the formal requirements model, and (c) model-based safety analysis should be applied. The work in [3] gave an idea of what might be achieved, but this paper has shown that POSE, in combination with Alloy, provides a process structure that directly supports these three goals. Also, [18] demonstrated that the derivation of a specification $S$ from $W, S \vdash R$, requires implementable requirements and as shown in [19], POSE supports this derivation through application of the PPT.

The combination of POSE and Alloy provide a structured process for producing a validated formal requirements model which can form a strong basis as the foundation of the rest of the development. By validating at each step in the requirements engineering process the risk of producing the wrong system is greatly diminished, and this is enhanced by producing a formal model that can be simulated and used to prove that it has the desired properties. Further, the same formal model can be used as the basis of the safety analysis, providing significant efficiency savings and again reducing risk.

The safety analysis use of HAZOPS and model analysis provides diversity and cross-checking. The results of using the different methods were comparable for this simple system, but the systematic nature of automated model analysis is likely to yield significant benefits on more complex systems where manual analysis could become error prone due to the amount of detail being handled.

Future work is based on three main themes: (a) to automate the failure injection process to improve the formal safety modelling capabilities; (b) further investigation into the composition problem, and to identify what help can be provided in POSE to support it; (c) to integrate the approach within an appropriate assurance case framework along the lines suggested in [16].

References

2. CCMB-2006-09-001: Common criteria for information technology security evaluation part 1: Introduction and general model. Ver. 3.1 Rev. 1 (September 2006)