ISSN 1744-1986

The Open University

Technical Report N° 2007/05

SIL4 process improvement with POSE and Alloy

Derek Mannering
Jon G Hall
Lucia Rapanotti

30th March 2007

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

http://computing.open.ac.uk

SIL4 process improvement with POSE and Alloy

Derek Mannering[1], Jon G. Hall[2], and Lucia Rapanotti[2]

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

Abstract. Safety Standards demand that industrial applications demonstrate they have the required safety integrity and this starts with the initial requirements phase. This paper shows how the Problem Oriented Software Engineering (POSE) framework, in conjunction with the Alloy formal method, supports this task through its ability to elaborate, transform and analyse the project requirements and thus develop a solution for an avionics case study. In particular, this work reports on how the POSE/Alloy combination was used in conjunction with the POSE safety pattern to improve the requirements analysis capabilities of an existing, successful safety critical development process. The results of applying this combination to an existing design showed that it could detect anomalies early in the life cycle that had previously been detected by much later (and more costly) validation work.

1 Introduction

Ensuring adequate safety is a crucial factor in the deployment of many embedded systems, including those used in avionics applications. This concern has been captured in safety standards such as the UK MoD's Defence Standard (DS) 00-56 [1] and the international IEC 61508 [2]. These standards require hazard identification and preliminary hazard analysis to occur in the early phases of the development process (e.g. [3]). This is consistent with studies that have shown that a large proportion of anomalies occur at the requirements and specification stages of a system development [4], [5]. A study by Lutz concluded that safety-related software errors arose most often from inadequate or misunderstood requirements [6]. Other work has highlighted the need to conduct a safety analysis of the requirements [7],[8]. These factors all support the notion that safety must be built into the design, with the evolving designs analysed to demonstrate that they have the desired safety properties [9] as early as possible in the life cycle—preferably during the requirements phase.

The first author is a member of his company's Mission System group, which has a successful DS 00-56 SIL4 capable process: briefly, system safety properties are formally captured in Z [10] and transcribed (rather than refined) into a detailed Z design specification, against which formal code proof using SPARK [11] is performed. An important validation step is the proof of conformance of the Z design specification against the formal Z safety properties. This process

provides a formal path from the high level safety properties down to the code that implements them. However, validation occurs well into the design process and can uncover anomalies with the Z safety properties.

The authors have previously shown how, within the Problem Oriented Software Engineering (POSE) framework [12], a safety pattern [13] can directly support the process of formulating a requirements model that is known to satisfy its identified safety requirements as required by the safety standards (e.g. [1]). The POSE safety pattern is efficient in that it uses the same information and models as used for the development task, and the overhead of having to validate specific safety analysis models is avoided. Previous applications of the POSE safety pattern have used structured textual requirements which were formalised into a Parnas Table-like form [14] for the safety analysis, including fault tree analysis (FTA) and functional failure analysis (FFA).

Although adequate for safety related applications, for SIL4 applications a higher integrity approach is desirable. In this paper, we therefore employ Alloy—a 'lightweight' formal notation that supports animation and formal proof of properties [15]—within the POSE safety pattern to examine an extant safety critical development, conducted using the SIL4 process sketched above in which anomalies in the Z safety properties were uncovered late in the process. The anomalies uncovered ranged from a mis-interpretation of the actual physical sequence (a modelling error), through to contradictory requirements. The ability to animate and check the POSE/Alloy model identified these anomalies. The goal of this paper is to show that, using the POSE/Alloy combination, those anomalies *could* have been identified and resolved earlier in the development life cycle. The ultimate goal of this work is to enhance and evolve what is already a successful SIL4 process, not to change to some novel process of unknown pedigree.

This accords with one of the fundamental goals of POSE research which is to identify and promote a move towards 'normal' software engineering design (Vincenti, [16]), thus avoiding the seemingly continual round of 'radical' design which plagues current software development practice [17].

The paper is organised as follows: background and related work are presented in Section 2. The POSE framework is briefly described in Section 3. Section 4 demonstrates the use of POSE and Alloy on a case study involving the development of requirements and high level architecture for a component of an aircraft warning system. Section 5 contains a discussion and conclusions.

2 Background and Related Work

The case study work presented in this paper is based on a multi-level safety analysis process typical of many industries. For example, commercial airborne systems are governed by ARP4761 [18]. ARP4761 defines a process incorporating Aircraft FHA (Functional Hazard Analysis), followed by System FHA, followed by PSSA (Preliminary System Safety Assessment, which analyses the proposed architecture). This paper is concerned with the latter, PSSA, but uses PSA (Preliminary Safety Analysis, a combination of hazard identification and pre-

liminary hazard analysis as required by the safety standards) in place of PSSA. In this paper requirements follow the fundamental clarification work of Jackson [19] and Parnas [14] which distinguishes between the given domain properties of the environment and the desired behaviour covered by the requirements. This work also distinguishes between requirements that are presented in terms of the stakeholder(s) and the specification of the solution which is formulated in terms of objects manipulated by software [20]. Therefore there is a large semantic gulf between the system level requirements and the specification of the machine solution. One of the goals of applying POSE is to bridge this gulf by transforming the system level requirements into requirements that apply more directly to the solution. However, there is a need to check that the transformed requirements can satisfy there safety obligations—hence the POSE safety pattern (Figure 1) was developed to include PSA as a 'continue or iterate back' gateway in the development process.

Alloy [15] is a lightweight formal method developed from the goal of combining the power of a SAT (Boolean satisfiability) solver with the descriptive power of the Z language. It has an active and growing user community and is continuously being developed and enhanced. It has strong animation and proof capabilities within well-defined limits, and allows complex behaviour to be modelled using clear, simple constructs. As such, it fits well as the modelling tool for the PSA part of the POSE safety pattern.

The POSE notion of problem used in this work fits well with the Parnas 4-Variable model, which has been used by Parnas *et al.* as part of a table driven approach [14]. This model and table-based approach is particularly well suited to defining embedded critical applications. This is demonstrated by the fact that they form the basis for the SCR [21], and the RSML methods. The RSML work led to the SpecTRM [22] methods, which form part of a human centred, safety-driven process which is supported by an artefact called an Intent specification [9]. Recent work in developing AMBERS [23], also uses the 4-Variable model (based on the SCR variant) and tables for the requirements phase and targets the SCADE [24] system for the subsequent development. AMBERS has similar goals to and is also compatible with POSE, but it does not include the specific high level PSA feasibility check.

The work of Anderson, de Lemos, and Saeed [7] share many of the principles and concepts that have driven the development of this work. Particularly the notions that safety is a system attribute and the need to apply a detailed safety analysis to the requirements specifications. The main advantages of the POSE approach over that work are: (a) it provides a framework for transforming requirements; (b) it is rich in traceability; and (c) the models it uses are suitable for the safety analysis. The latter means it is efficient because there is no need to develop 'new' models (with all its attendant validation problems) just to perform the PSA.

3 Problem Oriented Software Engineering

In POSE, software development is viewed as solving a problem, the solution being a machine—that is, a program running in a computer—that will ensure satisfaction of the requirement in the given problem world consisting of real-world domains. Typically the requirement concerns properties and behaviours that are located in the problem world at some distance from its interface with the machine. Like Problem Frames [25], POSE views the problem world as a collections of domains described in terms of their known, or *indicative*, properties, which interact through their sharing of phenomena, i.e, events, commands, states, etc.

POSE is defined as a formal Gentzen-style sequent calculus [26] that allows problems to be transformed into problems that are easier to solve, or that will lead to other problems that are easier to solve. A set of transformation rule schema defined in the calculus capture (atomic) discrete steps in development. Each requires a justification of application in order for the transformation to be solution preserving—simplifying only slightly, this means that a solution to a transformed problem is also a solution to the original problem—although justifications need not be formal. The combination of the justifications is an argument that the solution is adequate as a solution to the original problem. The interested reader is referred to [12] for a complete presentation of POSE.

In previous work [27] we have observed that POSE transformations combine to form a re-usable process template or 'pattern' for safety-critical development. One such process is shown in Figure 1 as a UML activity diagram. The activities in the figure include the following POSE activities:

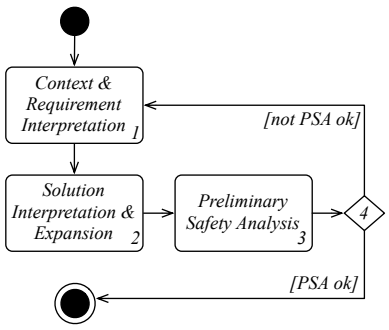

Fig. 1. POSE Safety Pattern

Context and Requirement Interpretation used to capture increasing knowledge and detail in the context (i.e., the environment into which the solution will be introduced) and requirement of the problem. Briefly, detail is added to a problem's context as knowledge of it grows;

Solution Expansion the choice and subsequent structuring of the solution according to a candidate solution architecture. Briefly, an architecture (logical and/or physical) for the solution is chosen, and used to transform the problem;

Preliminary Safety Analysis (PSA) a combination of problem simplification and traditional safety analysis conducted to ensure a feasible solution structure has been chosen.

The choice point (labelled 4) uses the outcome of the PSA to determine whether:

- the current architecture is viable as the basis of a solution; or
- whether backtracking and (re-)development of the problem (activity 1) and/or another candidate architecture (activity 2) should be chosen.

The pattern is iterative, ending when an architecture suitable for solution development is found. This process is iterative in that design choices, through the choice of candidate architecture, influence requirements, and vice versa. As we shall see, POSE allows the capture of many important other artefacts of the process, including a record of the choices that have been made, the rationale for the revision of requirements statements.

4 Case Study

The case study was concerned with examining the use of the POSE/Alloy combination in improving the front-end performance of a successful SIL4 process as applied to the design of an aircraft stores management system. As noted in the introduction, the original validation process correctly identified some subtle anomalies with the definition in Z of the high level safety properties, but these were found late in the process and lead to substantial changes. The role of this case study is to address two important questions:

(a) *could* these anomalies have been discovered earlier using POSE and Alloy, and
(b) *would* they have been discovered earlier by following a *reasonable* process based on POSE and alloy.

In this context, a reasonable process is one in which nothing special is done to identify the anomalies. We revisit these questions in the discussion of Section 5.

4.1 Stores Management System

For reasons of space only part of the case study is reported in this paper, namely the design of the selective jettison (SJ) functionality, i.e., the way in which stores are chosen for release from the aircraft. However, this part covers all the areas of interest. Figure 2 shows an aircraft as having six release stations—corresponding to Outer, Middle and Inner positions on each of the Starboard and Port wings. The pilot (P) controls the jettison of stores via the Selective Jettison Panel (SP),

also shown in Figure 2. The SP has (a) six selection switches, one for each station, (b) a three position SJ mode selection switch, (c) an 'In the Air' indicator lamp and (d) the SJ button which initiates the jettison.

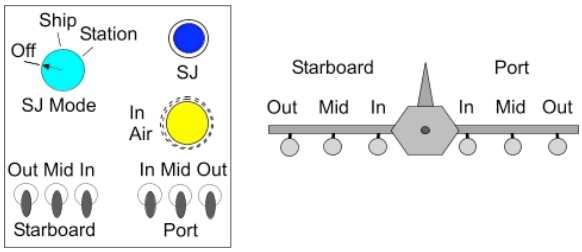

Fig. 2. Selective Jettison Panel (SP) & Aircraft schematic

4.2 Applying the POSE Safety Pattern

As the case study is based on an existing safety critical design many of the choices that comprise the design of the solution have already been made: a fuller description of the case study may revisit those choices, but we do not have the space to do so here. What follows is a short précis of how the POSE safety pattern is used to support later development:

The first step in the POSE safety pattern, Context & Requirement Interpretation (Figure 1[1]), involves understanding the existing formal and informal design information, information contained in the extant design. The high level safety requirements were captured as safety properties in the Z notation [10], the detailed safety requirements were defined using a Z operational model.

The second step in the POSE Safety pattern (Figure 1) is Solution Interpretation and Expansion. As we are replaying the successful deign the candidate solution architecture is already chosen as shown in Figure 3(a). It shows that each station has a Store Unit (SU) and a Suspension and Release Equipment (S&RE) associated with it. The SU provides the power to release or jettison the store held on the S&RE, under the control of the Safety Manager (SM) to be designed.

The third step of the POSE pattern is to perform a Preliminary Safety Analysis. As described above, this step combines problem simplification and traditional safety analysis. Problem simplification removes the non-direct domains (6x S&RE and P) and transform the requirements R at the system boundary to requirements RM that apply more directly to the required solution machine SM. The mechanism is explained in detail in [27] but is not covered in depth here.

[1] The formal notation of POSE reflects its genesis as a sequent calculus. For this paper, the formal notation is too unwieldy, and we use a graphical notation for convenience.

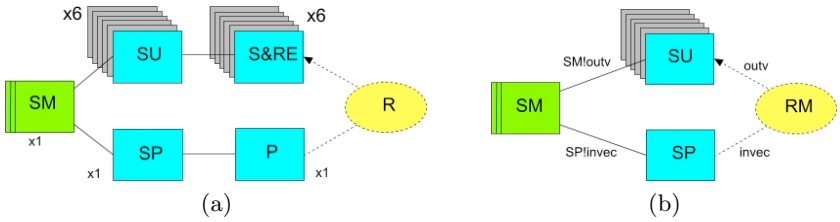

Fig. 3. (a)Stores Management System Architecture, including Store Units (SUs) and Suspense & Release Equipment (S&RE) for each station (x6), the Selective Jettison Panel (SP), the Pilot (P), Requirement (R) and the Safety Manager (SM) to be designed. (b) POSE Transformed Stores Management System Architecture

However, it is useful to provide an example of the type of transformation that occurs. One of the system level requirements from R is:

> The pilot shall select the SJ mode using the SJ Mode selection switch on the SP. SJ is inhibited if the SJ Mode switch is off.

The effect of the pilot on the system is, simply, to make their selection on the SP. After simplification of the problem by removing the Pilot, the requirement becomes:

> Given the SP status, SP shall have a SJ Mode selection switch which will indicate the selected SJ mode. SJ is inhibited if the SJ Mode switch is off'

This is part of the transformed requirements in RM (see Table 1). The transformed architecture is shown in Figure 3(b). To facilitate the analysis, the SM and the SP interface phenomena were selected to be the same as the requirements phenomena (outv and invec) with outv controlled by SM (indicated in the figure by writing SM!outv) and invec controlled by SP (indicated as SP!invec). These are defined as:

- outv : (i) `relp` Indicates a valid SJ sequence is in progress;
  (ii) `pulses` Defines which stations receive SJ pulses for this time instance;
  (iii) `balan` Indicates whether the SJ is to satisfy the balance condition.
- invec : (i) `mode` Indicates the SJ mode selection (off, Ship or station);
  (ii) `sels` Set defining the status of the SJ station selection switches;
  (iii) `sj` Indicates whether the SJ Button has been pressed;
  (iv) `fl` Indicates if the aircraft is in the air (air) or on the ground (grd).

The outv phenomena `balan` (indicating whether the balance algorithm has been applied to the SJ package) is not used by the SU.

Table 1. Transformed Requirements RM

Ref	Requirement Description
RM1	The SM shall ensure the safe release of stores from the S&RE when commanded by the Pilot via the panel, SP.
RM2	The SP shall have a Selective Jettison (SJ) Mode selection switch to allow selection of no SJ (Off), SJ of all stores (Ship) and station SJ.
RM3	The SP shall have six station selection switches, one for each station position. In Station SJ mode, a station must be selected before it can be jettisoned.
RM4	The SP shall have an In Air lamp, which will be lit if the aircraft is in the air.
RM5	SP shall have a SJ button to initiate the jettison and it must remain pressed for the duration of the jettison sequence. The following must hold prior to initiating the SJ: *CR*5.1 SJ mode selected (Ship or Station), *CR*5.2 Aircraft is in the air. *CR*5.3 If Station mode, the stations are selected on the SP.
RM6	The SM shall control the selection and jettison of stores from the SR&E, by interfacing to the SU to provide the conditioned power and drive signals to correctly drive and control the S&RE
RM7	(Safety) To mitigate inadvertent jettison hazards, SM shall not command the jettison of a store unless all the conditions of CR5 are satisfied.
RM8	The SM shall implement a Balance Algorithm which can remove items from a SJ package to ensure aircraft roll moments are contained within safe limits.
RM9	It shall be possible to SJ a store from a single station without regard to the aircraft balance algorithm calculation.
RM10	The SJ release sequence will be Out stores first, then Mid and then In.

The idea of reformulating requirements such as RM which apply directly to the required solution machine (SM in this case) is that it helps in the specification of the solution machine. However, before producing such a specification, it is important to establish that the requirements are feasible from a safety perspective; otherwise, the development must backtrack to develop a revised set of solution machine requirements. To establish requirements feasibility, POSE is used to reformulate them into a form suitable for safety analysis to the required safety level. In an earlier example, [27], Parnas Tables were used as the requirements were simple in form and the development context safety-related. In this case study, the requirements are more complex, and the ability to machine simulate and prove aspects of the formal model are much more desirable—hence the choice of Alloy for this purpose.

4.3 Formal Requirements Interpretation: the Alloy Model

The SJ sequence does not release all the stores at once, as this could result in a collision hazard, rather it releases stores in the step sequence `Out` $\rightarrow$ `Mid` $\rightarrow$ `In` with a delay between each step. This is captured by RM10 in Table 1 and the use of `cnt` in the Alloy model. That is, the SJ is not an atomic action, but rather a sequence of three atomic actions in sequence.

If a set of heavy stores were jettisoned from only one wing of the aircraft the aircraft would be subjected to a potentially lethal roll moment. To avoid this problem, jettison packages (including Ship) are subjected to a balance algorithm calculation, which can remove items from the SJ package to ensure that the aircraft remains within safe roll moment limits. This is captured as RM8 in Table 1. Store on Station (SoS) indications are used to signify if a store is present or not, these are used by the balance algorithm to make its calculations. However, it is recognised that these SoS indications could fail in such a way as to fool the balance algorithm. In particular, this might result in the balance algorithm inhibiting a jettison of a store when in fact jettisoning the store could result in a more benign residual roll moment. To counter this problem, the Pilot is allowed to SJ a single store, irrespective of the Balance algorithm. This is captured as RM9 in Table 1.

The requirements specification Alloy model of the SM was developed directly from the POSE model, using the same interface and phenomena as depicted in Figure 3(b). The functionality of this Alloy model was based on the POSE domain description information, the Z specifications, and the derived requirements, RM (Table 1). In Alloy terms [15], the SM module makes use of an extension to the standard Alloy Natural numbers utility file. The extension just involves defining terms for `Two` to `Ten`, in addition to the `Zero` and `One` already provided. There is not space to provide the full Alloy specification, so the following presents the main points of the model from which the full model could be recovered with additional reference to [15].

The signature (set) type definitions are based on an abstract set, with distinct single subsets. A typical definition for the type `SJMode` is shown below:

```
abstract sig SJMode {}
one sig  off, ship, stat extends SJMode {}
```

where `off` and `ship` correspond directly to there respective SJ mode switch settings and `stat` corresponds to station mode. Similar definitions were defined for (a) `SSet {son, soff}` to represent the SJ button and whether a valid jettison release pulse is allowed, (b) `StatsSel {s0, s1, s2, s3, s4}` where each `sn` covers a particular SJ switch station selection set, (c) `InAir {air, grd}` to represent the In Air indication, (d) `SPul {p0, p1, p2}` to represent the jettison release pulse patterns, and (e) `BAL {nob, no_bal, is_bal}` to represent if the SJ package is required to satisfy the balance algorithm. Where `nob` represents not applicable, `no_bal` represents balance algorithm not applied and `is_bal` means the balance algorithm is applied to the SJ package. These types are used to define the overall state of the SM as follows:

```
sig SMState {   ntime : Natural,
                mode  : SJMode,             relp   : SSet ,
                sj    : SSet,               pulses : SPul,
                sels  : StatSel,            balan  : BAL,
                fl    : InAir,              cnt    : Natural } {}
```

In `SMState` `ntime` represents the system simulation time reference and is modelled as a natural number, and `cnt` is a local time counter used for timing the SJ release sequence. Effectively, the other state components on the LHS are

inputs from the SP and were defined using the SP! invec phenomena from the POSE model. The other RHS state components are outputs to the SU defined using the SM! outv phenomena. The `relp = son` means it is valid to provide SJ pulses, and `pulses` defines the pattern of pulses to be output at this time (note `p0` means no pulses output). The initial conditions are defined by the predicate init.

```
Pred init (sm:SMState) {sm.mode=off && sm.sels=s0 && sm.sj=soff &&
  sm.fl=grd && sm.ntime = Zero && sm.relp = soff && sm.pulses = p0 &&
  sm.balan = nob && sm.cnt = Zero}
```

The trace model used is based on that used in Chapter 2 of [15] and involves using a linear ordering to order the traces with init as the first trace. In the following the `InVec(Zero/stat/s0/soff/grd)` represents the input vector to the SM and is short for the predicate `InVec(sm.ntime,One,m,stat,sel,s0,s,soff,f,air)`. `Stim()` represents the simulation time increment predicate and is short for `Stim(sm,sm')`. The trace timing simulation covers eight slots from `Zero` through to `Seven`.

```
Fact traces {
 init(sms/first())
 all sm:SMState - sms/last() | let sm' = sms/next(sm) |
  some m:SJMode, sel:StatSel, s:SSet, f:InAir |

(InVec(Zero/stat/s0/soff/grd)  && Stim() && SMfun(sm,sm',m,sel,s,f)) or
(InVec(One/stat/s0/soff/air)   && Stim() && SMfun(sm,sm',m,sel,s,f)) or
(InVec(Two/stat/s1/soff/air)   && Stim() && SMfun(sm,sm',m,sel,s,f)) or
(InVec(Three/stat/s1/son/air)  && Stim() && SMfun(sm,sm',m,sel,s,f)) or
(InVec(Four/stat/s1/son/air)   && Stim() && SMfun(sm,sm',m,sel,s,f)) or
(InVec(Five/stat/s1/son/air)   && Stim() && SMfun(sm,sm',m,sel,s,f)) or
(InVec(Six/ship/s1/son/air)    && Stim() && SMfun(sm,sm',m,sel,s,f)) or

(nat/gte(sm.ntime, Seven) && m=off && sel=s0 && s=soff && f=air   &&
SMfun(sm,sm',m,sel,s,f)) }
```

The behaviour of the SM is defined by the predicate `SMfun`. This sets the valid SJ pulse indicator (`sm'.relp=son`), checks balance (`balance(sm')`) and outputs SJ pulses in sequence (`pulTab(sm')`) if a valid SJ button press is detected or the SJ sequence is still valid and continues. Otherwise, it resets the SM state to a safe value.

```
pred SMfun (sm,sm':SMState, m:SJMode, sel:StatSel, s:SSet, f:InAir) {
(sm'.mode = m  && sm'.sj = s && sm'.fl = f && sm'.sels = sel) &&

(((sm.mode = stat && sm.sels != s0 && sm.sj = soff && sm.fl = air &&
   m = stat && sel = sm.sels && s = son && f = air)
    => sm'.relp = son && sm'.cnt = One && pulTab(sm') && balance(sm'))
&&

 ((sm.mode = stat && sm.sels != s0 && sm.sj = son && sm.fl = air &&
   sm.relp = son && m = stat && sel = sm.sels && s = son && f = air)
   => sm'.relp = son && sm'.cnt = incnt(sm.cnt) && pulTab(sm') &&
      balance(sm')) &&

(not(m = stat && sel != s0 && sel = sm.sels && s = son && f = air)
  => sm'.relp=soff && sm'.cnt=Zero && sm'.pulses=p0 && sm'.balan=nob)
       ) }
```

The function `incnt()` increments `cnt`, except if `cnt` is `Three` when resets it to `One`. The predicate `PulTab()` is used to define the SJ pulse sequence timing, with `sm.cnt=One` corresponding to SJ from the Out stations, `Two` from Mid and so on. By varying the content of the `s1` to `s4` definitions, the complete range of SJ package selections can be covered.

```
pred pulTab (sm : SMState) {
/* s1 -- Port Out & Starboard In selected for SJ */
       (sm.sels = s1 && sm.cnt = One => sm.pulses = p1) &&   //Out
       (sm.sels = s1 && sm.cnt = Two => sm.pulses = p0) &&   //Mid
       (sm.sels = s1 && sm.cnt = Three => sm.pulses = p1) && //In

*** Similar SJ package definitions for S2, S3 and S4 ***

 /* s0 -- No stores selected for SJ */
       (sm.sels = s0 => sm.pulses = p0)             }
```

The balance algorithm is covered by the predicate `balance()`, where if there are one or zero SJ pulses, then no balance (no_bal) otherwise balance is required.

```
pred balance(sm : SMState) { sm.relp = son &&
((sm.sels=s0 or sm.sels=s4) => sm.balan = no bal, sm.balan = is bal ) }
```

4.4 POSE Safety Pattern Step 3: PSA

Having produced the formal Alloy model of the requirements RM the next step is to validate that the model meets the requirements, and to then perform the PSA. A variety of simulation runs were used to validate the model. Each run used a different combination of `InVec()` values to explore the behaviour of the model over its range of inputs. The results validated that the model satisfied RM. The PSA followed the pattern used on earlier examples (e.g. [13]), with Functional Failure Analysis (FFA) [18] being applied to identify any system issues. This paper concentrates on the functionality issues and does not consider the system hardware analysis. An important FFA issue was 'Multiple station SJ release but balance not applied'. Simulation of the Alloy model using the `s1` station selection set (refer to InVec() definition above) indicated that `balan = no_bal` for this set. However, `s1` is a multiple SJ—Port Out and Starboard

In, so the release pulse sequence should be `p1;p0;p1` for `cnt` of `One`, `Two` and `Three` respectively—so balance should be applied. Simulating with other station selection sets indicated that balance was only applied when more than one pulse was required at a particular time (i.e. `p2`) and only for that time. The required behaviour is that balance should always be applied unless the SJ package consists of a single release (a single `p1` with two `p0`s). The problem occurs in the high level Z specification of the SM, which contains the term:

$$(\#releaseLocations > 1 = \theta \mathsf{SM} \in \mathsf{balanced})$$

where # means the cardinality of the set and θSM means the bound values of SM. Now `releaseLocations` is the number of jettison pulses being applied—this is modelled in Alloy by the function `balance()`, which specifies no balance only if there are zero (`p0`) or a single (`p1`) pulse. The problem is that (as noted above) SJ is not an atomic action, but rather three: one each for Out, Mid and In. The functionality given above ensures each atomic action satisfies the balance, but does not ensure the entire SJ sequence does. Hence selection set `s1` which results in the pulse sequence `p1;p0;p1` meets the criteria for each of its atomic components, indicating that the model does not require it to be balanced, when it should be. The solution is to base the balance calculation on the SJ station selection and not the jettison pulses. Implementing this change, results in `balance()` being updated as follows:

```
pred balance(sm : SMState) { sm.relp = son &&
((#sm.sels > 1 => sm.balan = no_bal, sm.balan = is_bal ) }
```

Simulating with this form resulted in the correct behaviour for `s1` and the other selection set combinations. Aspects of this behaviour were then proved.

Other FFA issues were addressed in a similar fashion, and the modelling identified the further known anomaly and a not previously identified one. The latter concerned an inconsistency between two of the Customer supplied requirements documents. In all these cases the implementation was checked and it was confirmed that it operated as intended. That is, the problem was with the requirements specification.

5 Discussion and Conclusions

The case study work agrees with the results obtained from [27,13] and again demonstrates that the POSE safety pattern successfully allows the safety feasibility of a system's requirements to be analysed early in the development life cycle. As in the earlier case work, the analysis was found to be quick and efficient since it uses the same information and models that are produced to support the development work. Not having to produce and validate a special model for the safety work was a significant advantage.

Earlier papers ([12,27,13]) have shown that POSE is flexible enough to work well with a variety of common development approaches. The results of this case study further confirm this finding by showing that the POSE/Alloy combination

works well and is suitable for supporting the front-end work of a SIL4 process. This ability to work with existing development processes is important, because it supports the evolution and improvement, rather than the replacement, of these processes. This is in line with the goal of improving the 'normal' design, and avoids the damaging 'radical' design cycle.

The task of using the POSE/Alloy combination to improve the front-end of the SIL4 process asked two major questions. The first, and easiest to address, was could the POSE/Alloy combination detect the anomalies. The results from Section 4.4 demonstrate that both POSE and Alloy can be used in combination to detect the anomalies of interest. The second question is would the POSE/Alloy combination have detected these anomalies. That is, would the anomalies have been discovered by following a reasonably expected process? This is more difficult to answer because it is a process question. A plausible answer can be gleaned from Section4, which shows that POSE and Alloy were used as recommended and found the anomalies of interest. This imparts a high degree of confidence that the process used with the POSE/Alloy combination would be able to detect these and similar anomalies, and at an early phase in the development life cycle. Therefore technically the POSE/Alloy combination could and would be expected to provide the required process improvement for the SIL4 process. The next step is to evaluate the process improvement on a project using normal, but suitably trained, project engineers rather than those specialising in the POSE and Alloy techniques, to see if similarly encouraging results are obtained.

The use of POSE is being explored in many contexts. This and companion papers work within safety critical system engineering design. Other areas include those of security- and mission-critical, as well as new, as yet unpublished, work in drug design. It is encouraging to see the potential breadth of POSE ideas, as it is to see its potential for combining with other development practices on the detail of development.

Acknowledgements

We are pleased to acknowledge the financial support of IBM, under the Eclipse Innovation Grants. Thanks also go to our colleagues in The Open University, especially Michael Jackson.

References

1. DS0056-3: Defence Standard 00-56 issue 3 (1997) Safety Management Requirements for Defence Systems.
2. IEC: IEC Standard 61508 (March 2000) Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems.
3. Martino, P.A., Muniak, C.: The role of system safety engineering in product safety. In: 20th International System Safety Conference. (2002)
4. Ellis, A.: Achieving safety in complex control systems. In: Safety Critical Systems Symposium. (1995)

5. Leveson, N.: Safeware: System Safety and Computers. Addison-Wesley Publishing Company (1995)
6. Lutz, R.R.: Analysing software requirements errors in safety-critical embedded systems. In: IEEE International Symposium Requirements Engineering. (1993)
7. de Lemos, R., Saeed, A., Anderson, T.: On the integration of requirements analysis and safety analysis for safety-critical systems. Technical report, University of Newcastle upon Tyne (1998)
8. A. Gerstinger, G.S., Winkelbauer, W.: Safety versus reliability: Different or equal. In: 20th International System Safety Conference. (2002)
9. Leveson, N.G.: Intent specifications: An approach to building human-centered specifications. IEEE Transactions on Software Engineering **26** (2000) 15–35
10. Spivey, J.M.: The Z Notation: A Reference Manual. 2nd edn. Prentice-Hall (1992)
11. Barnes, J.: High Integrity Ada – The SPARK Approach. Addison-Wesley (1997)
12. Hall, J.G., Rapanotti, L., Jackson, M.: Problem-oriented software engineering. Technical Report 2006/10, Department of Computing, The Open University (2006)
13. Mannering, D., Hall, J.G., Rapanotti, L.: Towards normal design for safety-critical systems. In Dwyer, M.B., Lopes, A., eds.: Proceedings of FASE 2007. Volume 4422 of Lecture Notes in Computer Science., Springer Verlag Berlin Heidelberg (2007) 398–411 To appear.
14. Courtois, P.J., Parnas, D.L.: Documentation for safety critical software,. In: 15th International Conference on Software Engineering. (1997)
15. Jackson, D.: Software Abstractions Logic, Language, and Analysis. The MIT Press (2006)
16. Vincenti, W.G.: What Engineers Know and how they know it: Analytical studies from Aeronautical History. The Johns Hopkins University Press (1990)
17. Maibaum, T.: Mathematical foundations of software engineering: a roadmap. Presentation at ICSE 2000, 7-9 June (2000)
18. SAE: ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment (1996)
19. Zave, P., Jackson, M.A.: Four dark corners of requirements engineering. ACM Transactions on Software Engineering and Methodology **6**(1) (1997) 1–30
20. van Lamsweerde, A.: Requirements engineering in the year 00: A research perspective. (2000)
21. Bharadwaj, R., Heitmeyer, C.: Developing high assurance avionics systems with the scr requirements method. In: 19th Digital Avionics Systems Conferences. (2000)
22. Leveson, N.G.: Completeness in formal specification language design for process-control systems. In: Proceedings of the third workshop on Formal methods in software practice. (2000)
23. da Cruz, M.F., Raistrick, P.: AMBERS: Improving Requirements Specification Through Assertive Models and SCADE/DOORS Integration. In: Proceedings of the Safety Critical Systems Symposium. (2007)
24. Esterel: Getting Started with SCADE. (2007) http://www.esterel-technologies.com/technology/getting-started/scade.
25. Jackson, M.A.: Problem Frames: Analyzing and Structuring Software Development Problems. 1st edn. Addison-Wesley Publishing Company (2001)
26. Kleene, S.C.: Introduction to Metamathematics. Van Nostrand, Princeton, NJ. (1964)
27. Mannering, D., Hall, J.G., Rapanotti, L.: Relating safety requirements and system design through problem oriented software engineering. Technical Report TR2006/11, Computing Department, The Open University (2006)