Conferences in Research and Practice in Information Technology

Volume 156

Australian System Safety Conference 2014

Australian System Safety Conference 2014

Proceedings of the
Australian System Safety Conference (ASSC 2014),
Melbourne Australia, 28th-30th May 2014

Tony Cant, Eds.

Volume 156 in the Conferences in Research and Practice in Information Technology Series.
Published by the Australian Computer Society Inc.

Published in association with the ACM Digital Library.

Australian System Safety Conference 2014. Proceedings of the Australian System Safety Conference (ASSC 2014), Melbourne Australia, 28th-30th May 2014

Conferences in Research and Practice in Information Technology, Volume 156.

Editors:
Tony Cant

6 Landowne Tce. Walkerville SA 5081
E-mail: canttony@gmail.com

Series Editors:
Vladimir Estivill-Castro, Griffith University, Queensland
Simeon Simoff, University of Western Sydney, NSW
crpit@infoeng.flinders.edu.au

Publisher: Australian Computer Society Inc.
PO Box Q534, QVB Post Office
Sydney 1230
New South Wales
Australia.

Conferences in Research and Practice in Information Technology, Volume 156
ISSN 1445-1336
ISBN 978-1-921770-39-5

Printed November 2014 by Griffith University, CD proceedings.

Table of Contents

Preface

The *Australian System Safety Conference 2014* was held at the RACV City Club, Melbourne, on 29-30 May, 2014. The conference, jointly sponsored by the Australian Safety Critical Systems Association (aSCSa) and the Australian Chapter of the System Safety Society (SSS), had the theme: "Software Safety: New Challenges and Solutions" and was attended by more than 70 participants. This year, the organising committee was delighted to have the participation of four keynote speakers:

– Prof Dines Bjorner (Professor Emeritus of Computer Science, Technical University of Denmark)
– Prof Nancy Leveson (Professor of Aeronautics and Astronautics and Professor of Engineering Systems, Massachusetts Institute of Technology, USA)
– Prof Peter Lindsay (Boeing Professor of Systems Engineering, University of Queensland, Australia)
– Dr Colin O'Halloran (Department of Computer Science, University of Oxford and Director, D-Risq Ltd, UK)

A program of tutorials was also held prior to the conference. Full program details are available from `asssc.org/conf2014`. More information on the aSCSa can be found at `www.safety-club.org.au`.

The Organising Committee is very grateful to the authors for the trouble they have taken in preparing their work to be included in these conference proceedings. The papers were peer-reviewed for relevance and quality by the Program Committee. Note, however, that the views expressed in the papers are the authors' own, and in no way represent the views of the editor, the Australian Safety Critical Systems Association, the System Safety Society, or the Australian Computer Society. The fact that the papers have been accepted for publication should not be interpreted as an endorsement of the views or methods they describe, and no responsibility or liability is accepted for the contents of the articles or their use.

The committee also wishes to thank the conference sponsors for their support: the Australian Computer Society; Adacore; RGB Assurance; Nova Systems and the Defence Materiel Organisation in the Australian Government Department of Defence. These organisations have all helped to make the conference a success.

I wish to thank all those involved in organising the conference (listed below). In particular, I would like to acknowledge the commitment and drive of my colleagues Clive Boughton, B.J. Martin, Kevin Anderson, Holger Becht, Alan Coxson and Tariq Mahmood, who worked hard to make sure that the conference was a success.

We are also grateful to Anthony Ellard of the Australian Computer Society for his administration of the web-based registration system, as well as the Melbourne Branch of the ACS for their assistance. Finally, our thanks to the Computer Systems and Software Engineering Board of the ACS for ongoing support.

Tony Cant,
Organisers of ASSC 2014
November, 2014

Programme Committee

Keynote Papers

Nose-Gear Velocity – A challenge problem for software safety

Colin O'Halloran
Department of Computer Science
Oxford University
Wolfson Building, Oxford OX1 3QD, UK

colin.ohalloran@cs.ox.ac.uk

Abstract

Developing a safety critical cyber-physical system using conventional software development methods is time consuming and expensive. The reason why it is difficult is because of the need to produce evidence of assurance that is based solely on testing. This paper addresses this problem by exploiting simulation and automated formal methods that are hidden behind popular commercially available model based software development.[1]

Keywords: Model Based Development, Automated Verification, Proof, Simulink®, Software Requirements, Cyber Physical System.

1 Introduction

A safety requirement for software that interacts with the physical world will at some point need to be compared against physical values. This means that the requirement implicitly makes assumptions about how the software will "perceive" and "affect" its physical environment. The first difficulty is that the laws of physics cannot be made to conform to what is convenient to the software system, unlike the flexibility a human can exhibit when interacting with software systems.

A second difficulty is that it is often difficult to develop the specification of a software system that meets a safety requirement because the physical components to which it is connected can be subject to many physical variables. The task of developing an appropriate specification is further complicated by the fact that the physical components are subject to failure.

Jones, Hayes and Jackson (2007) propose an approach to address this problem - "Our method is conceptually simple: we ground our view of a desired computer system (or "silicon package") in the external physical world. This is the *problem world* whose phenomena are to be measured and influenced by the overall system. Having agreed with the customer the desired behaviour in the problem world, we record –and again obtain conformation of acceptability– assumptions about the physical components outside the computer itself. Only then do we *derive* the specification of the software to run in the computer."

In other words the proposed approach is to first specify the requirements of the overall system in the physical world and then to determine necessary assumptions about components of that physical world. Only after this is done is a specification of the computational part of the control system derived.

Such an approach is novel to software engineering but is recognisable in the domain of control system engineering where differential equations are derived in conjunction with a model of the physical world. The novelty for control system engineering would be the derivation of a specification from which could be given a formal proof that, subject to the given modelling assumptions, an implementation meets the specification.

The novelty of the approach proposed described in this paper is that it combines the approach of control system engineering with that of software engineering. In particular it builds upon the approach described by Jones, Hayes and Jackson (2007) in that it enables a more natural operational way of capturing assumptions about physical phenomena.

To illustrate the approach a challenge problem concerning the estimation of the ground velocity of an aircraft is used. The challenge problem was set as part of a theorem proving in certification workshop (Gordon 2013). The requirement that the software must satisfy is that

"the estimated Ground Velocity shall be within 3 km/hr of the true velocity of the aircraft at some moment within the past 3 seconds".

The full details of the example are given by Joyce (2011).

The seemingly simple requirement leads to some subtle issues about the physical environment that the software must deal with. For example what is meant by the *true velocity* of the aircraft in a specification before the software has been tested on an aircraft?

2 Details of the illustrative example

2.1 Further details of the example

In the example (Joyce 2011) the velocity is estimated by measuring the elapsed time of a full rotation of the nose gear wheel. Each time this wheel completes a full rotation, a pulse (informally called a "click") is generated by an electro-mechanical sensor connected to a computer.

[1] Copyright © 2014, Australian Computer Society, Inc. This paper appeared at the Australian System Safety Conference (ASSC 2014), held in Melbourne 28-30 May, 2014. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 156, Ed. Tony Cant. Reproduction for academic, not-for profit purposes permitted provided this text is included.

The pulse causes a hardware interrupt which is serviced by an interrupt routine that increments a 16-bit counter called "NGRotations" and stores the current time in a 16-bit variable called "NGClickTime".

The Real-time Operating System (RTOS) periodically invokes another function that uses the current value of this counter to re-calculate an estimate of the current velocity and store the result in a variable called "estimatedGroundVelocity". The RTOS ensures that this update function is invoked at least once every 500 milliseconds; however, the exact timing of each invocation relative to a hardware interrupt is not predictable.

In addition to the counter that records rotations of the wheel, this update function has read-only access to a 16-bit counter called "Millisecs" which is incremented once every millisecond. This counter is the same source of time used by the interrupt routine to update NGClickTime. The circumference of the nose gear wheel is also available to the update function as the value of a compile time constant called "WHEEL_DIAMETER". The update function may store private data values that are protected from invocation to invocation. An example of this calculation is shown below in Figure 1 (Joyce 2011).

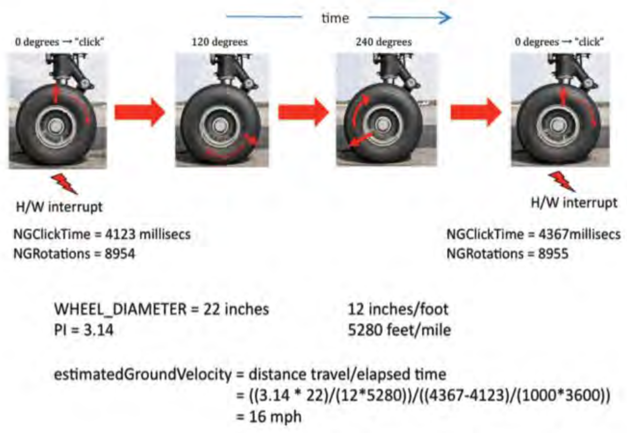


Figure 1

In this particular example, it is assumed that NGRotations has just been incremented when the calculation is performed, i.e. the update function is invoked within a few milliseconds after a new 'click'. However, the implementation of this function should handle the more general case in which the update function can be invoked at any time.

2.2 Experimenting with a specification

The initial simplicity of the problem belies the potential for some considered thinking. In the physical world tyres will slip and skid. The nose-gear tyre will compress under the load of the aircraft changing the effective radius of the tyre as the load it carries increases. These couple of physical world issues will affect the estimated velocity.

If the aircraft is landing then wheel rotation before it is in contact with the ground needs to be accounted for, perhaps with a weight on wheels signal. There are about a dozen other assumptions that need to be made, e.g. the maximum acceleration of the aircraft on the ground, the amount of variability in the timing of the pulse generated by the electro-mechanical sensor connected to the nose gear wheel.

As observed in the original statement of the problem, the requirements specification is not precise, e.g., what does "accurate" mean? What about when the aircraft is being pushed backwards – is this a negative velocity?

Clearly the questions about the physical environment will affect the specification of the software if it is to meet the simply stated requirement about the accuracy of the estimated velocity. In the approach proposed by Jones, Hayes and Jackson (2007) such questions are recorded and the necessary assumptions required by the specification are agreed with a customer and then recorded.

It is the thesis of this paper that identifying such assumptions is difficult and requires an experimental phase to provide evidence that is meaningful to a customer in order to agree environmental assumptions. Implementing a conjectured specification in order to carry out experiments on a real aircraft is expensive. Instead the use of an appropriate modelling and simulation environment enables computer based experimentation to take place before later validation on a test rig or an aircraft. The approach taken in this paper is like that of Jones, Hayes and Jackson (2007) in that it focuses on the physical environment first in order to shape the specification of the software, but through modelling the environment in the MathWorks®' modelling language of Simulink®. An algorithm to interact with the environment can then be developed also in Simulink® and experiments conducted before code is automatically generated from the algorithm.

The capability to conduct experiments on a host machine before generating the software provides more chance of getting the right specification of the software. The capability of automatically generating the software means that later validation testing on an aircraft leads to a rapid cycle of: refining the model of the physical environment; adjusting the algorithm; regenerating the code and then running **validation** testing again. The verification that the code implements the Simulink® specification is discussed later.

3 The Nose-Gear Problem

3.1 A Simulink® Model

3.1.1 Model Overview

The composition of models in Simulink® is shown in Figure 2 below.

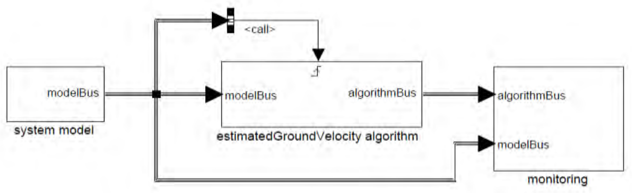


Figure 2

The model of the relevant parts of the landing gear system is labelled “system model” in Figure 2. The values that it generates are communicated over a bus to the algorithm for estimating the ground velocity and a monitoring system for capturing the outputs of the simulation as it happens.

3.1.2 The Physical System Model

The model of the dynamics of the nose-gear system, labelled “system model” in Figure 2, is shown in Figure 3.

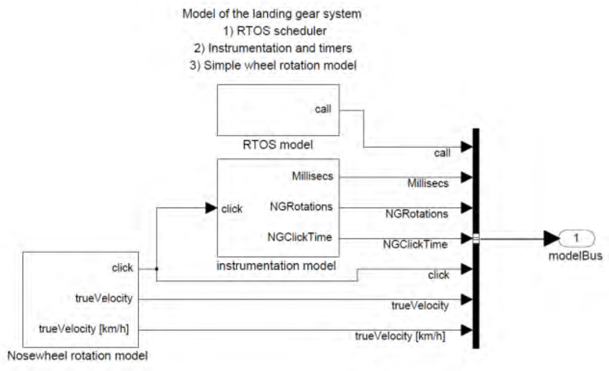


Figure 3

The block labelled RTOS in Figure 3 ensures that the algorithm for estimating the ground velocity is invoked at least once every 500 milliseconds. It incorporates some randomness so that the exact timing of each invocation relative to the hardware interrupt is not predictable, as required by Joyce (2011).

The block labelled “instrumentation model” in Figure 3 takes as input the signal from the electro-mechanical sensor that a full rotation of the nose-gear wheel has occurred. The outputs of the block “instrumentation model” are: the current incremented value of a 16-bit counter called “NGRotations” by Joyce (2011); the current stored time from a 16-bit variable called “NGClickTime”; and the current time in milliseconds.

The block “Nose-Wheel” rotation contains the relevant physical model of the wheel.

3.1.3 Nose-Wheel Rotation Model

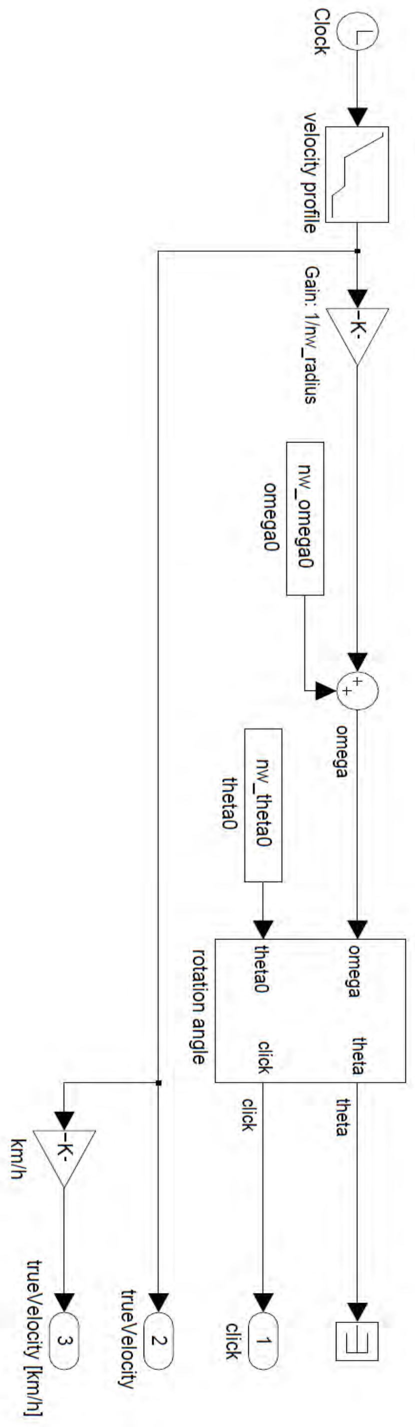


Figure 4

The model of the wheel is shown below in Figure 4. The model is clocked and uses this to take values from a velocity profile of the aircraft’s true velocity at different points in time. The true velocity value is in miles per hour; therefore it is necessary to convert it into kilometres per hour (because the software requirement is in km/hour) before being output from the block in Figure 3.

Wheel rotation before the wheel is in contact with the ground is taken into account with the block “nw_omega0”. The block “nw_omega0” assumes a weight on wheels signal to differentiate between rotations on the ground and off the ground.

The initial displacement of the wheel from the point where the electro-mechanical signal is generated is accounted for by “theta0”. The signals “omega” and “theta0” are inputs to the block labelled “rotation angle” in Figure 4. The subsystem corresponding to “rotation angle” is shown below in Figure 5.

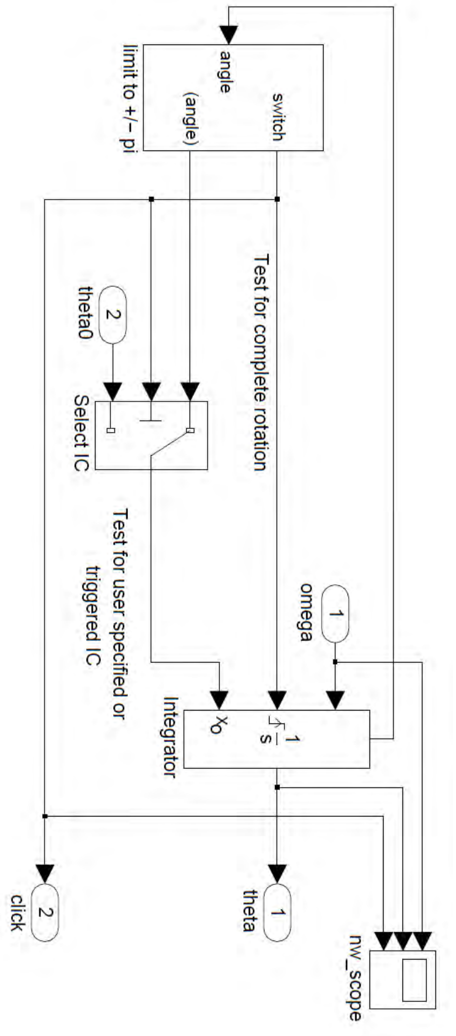


Figure 5

The model in Figure 5 essentially outputs the accumulated angle the nose-gear wheel has rotated through and the “click” that denotes a full rotation of the wheel. Note that the integrator block in Figure 5 is operating in the continuous domain unlike the algorithm for estimating the velocity which operates in the discrete domain.

Although the model is simple it is sufficient to develop a reasonable estimation algorithm and it demonstrates the principle of developing an operational like specification of the physical environment.

3.2 The Low Level Requirements

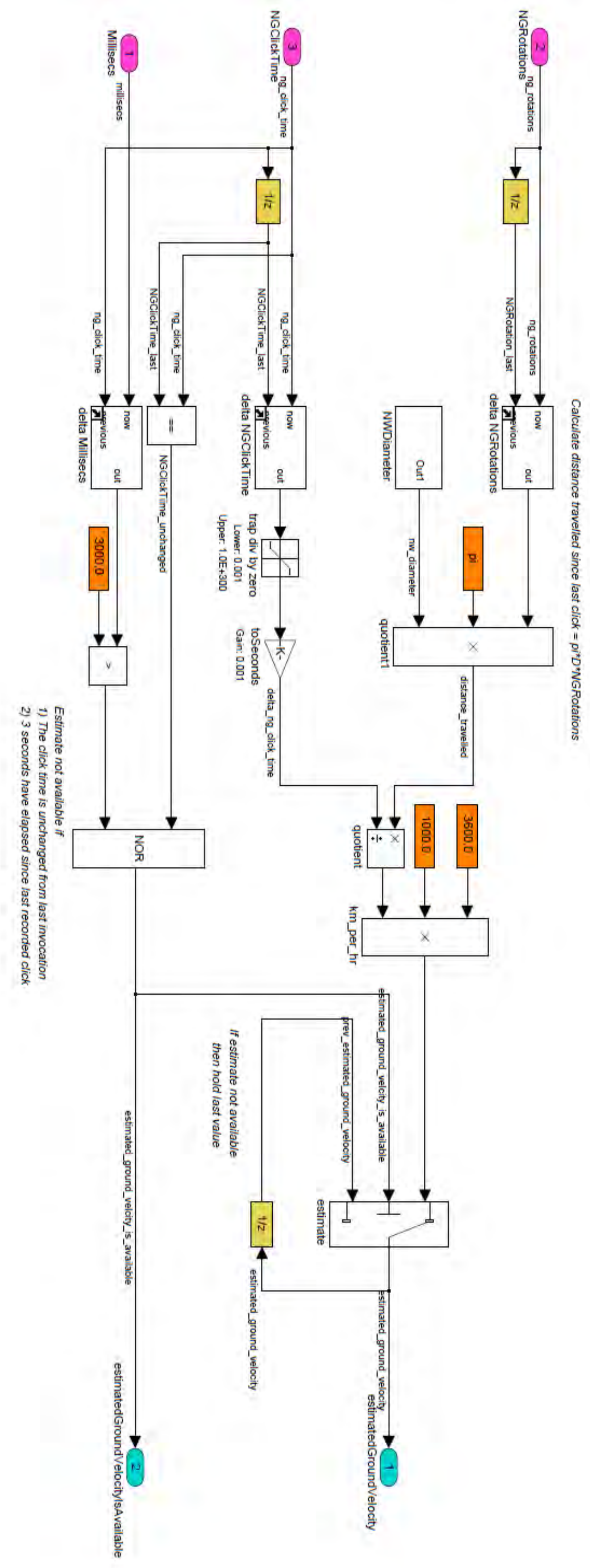


Figure 6

Figure 6 is a discrete model, of the estimation algorithm, from which code can be automatically generated. An earlier version of the algorithm might have operated in the continuous domain before being discretised. In any case some transformations of the model will be necessary to provide information necessary for an automatic code generator to do its job. In the approach advocated in this paper, further constraints will be necessary in order to also make it automatically verifiable. None of these constraints should be unduly onerous and be akin to those kinds of constraints you might find in a normal software development policy.

Running an initial model of the estimation algorithm with the physical model, as well as the version ready for automatic code generation enables validation of the low level requirements through experimentation via simulation on a host computer.

3.3 Experimentation

Using the model of the physical environment described in section 3.1 an algorithm that is conjectured to estimate the ground velocity of the aircraft can be measured against the true velocity, also contained in the physical model.

Examples of plots for the algorithm are shown below in Figure 7 and Figure 8. All plots show a parameter plotted against time which runs for 120 seconds.

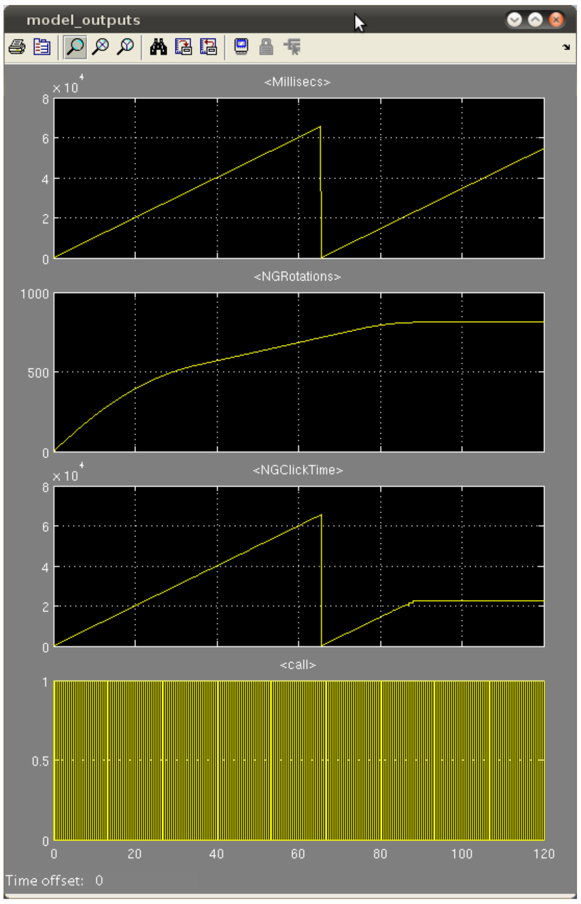


Figure 7

The first plot of Figure 7 labelled <Millisecs> is the clock time in milliseconds, which mimics the 16-bit behaviour described by Joyce (2011). The second plot <NGRotations> is of the counter, incrementing each time the wheel passes through 0^{o}. This also emulates 16-bit behaviour and if left to run long enough this would overflow or wrap around.

The third plot labelled <NGClickTime> is the time in milliseconds at which the latest click was detected, again emulating 16-bit behaviour. The final plot labelled “call” is a simulation of the interrupt generated by the RTOS to trigger a call to the estimation algorithm. By default this goes high at 500ms intervals. Adjusting workspace parameters in the simulation will introduce some variation in this so that calls at less than 500ms intervals can also be examined. Units are binary and the algorithm is called using a rising edge trigger, i.e. at the point when the signal goes from 0 to 1.

The second set of plots is shown in Figure 8 below.

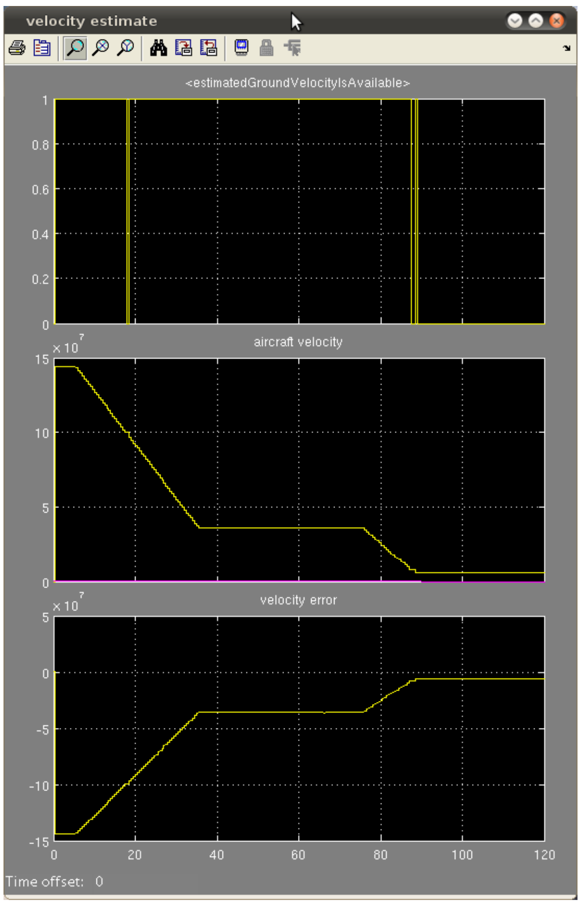


Figure 8

The first plot shows when the estimate for the ground velocity is valid; the units shown are binary. The second plot of aircraft velocity is a composite plot showing the true velocity (along the bottom) and the estimate (decreasing), the units are kilometres per hour. The final plot is of velocity error, i.e. the error between the estimate and the true value. This starts very high because within the 500ms of simulated time, the algorithm has yet to be called and the initial estimated speed is zero but the true velocity is 144 km/h. An improvement to the way the plotting is done would be to take into account the validity flag because zooming in on the graph (using MATLAB’s magnifying glass tool and rubber banding with the mouse) shows that the flag is low during that time.

In any case the point is that there is considerable scope for carrying out experiments to establish the validity of the algorithm before any coding is done. The validation is informal, but provides the basis for a later formalisation which has a much greater chance of successfully proving the software requirement.

4 Automatic Code Generation and Automatic Verification

Using the commercial auto-coder called B4S, from Applied Dynamics International (ADI 2014), the Simulink® model in Figure 6 was used to generate Ada code within a predictable subset that has a formal semantics.

A tiny fragment of the output of the B4S auto-coder is shown in Figure 9 below.

```
--  Saturate:
nosegear_velocity/estimatedGroundVeloc
ity/trap div by zero
--
-- # witness
"nosegear_velocity/estimatedGroundVelo
city/trap div by zero//In/1"
<--- "[TMP4]"
-- #
if ( tmp4 < 0.001 ) then
tmp5 := 0.001;
elsif ( tmp4 > 1.0E+300 ) then
tmp5 := 1.0E+300;
else
tmp5 := tmp4;
end if;
-- # witness
"nosegear_velocity/estimatedGroundVelo
city/trap div by zero//Out/1" <---
"[TMP5]"
-- #
```

Figure 9

Among the usual comments generated by an auto-coder are special annotations that record traceability information. For example the annotation "-- # witness" is followed by an address in the Simulink® model from which the code was generated.

For the fragment of code in Figure 9 the address refers to the block shown in Figure 10 that also appears as part of the model in Figure 6. The annotation refers to the top level model nosegear_velocity (that is shown in Figure 2) that contains the model estimatedGroundVelocity (that is shown in Figure 6) which in turn contains the block "trap div by zero".

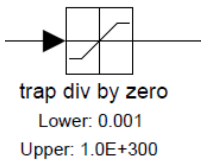


Figure 10

The suffixes "//In/1" and "//Out/1" refer to the input and output signal to the block shown in Figure 10. The witness annotations states that the input to the block is represented by the variable TMP4 and the output by TMP5. The implementing code is the conditional shown in Figure 9.

Using the traceability information, represented as witness annotations generated from the auto-coder, the CLawZ toolset (O'Halloran 2013), can automatically verify the generated code against a Z representation of the Simulink® model. A representation of the toolset is shown below the line in Figure 11.

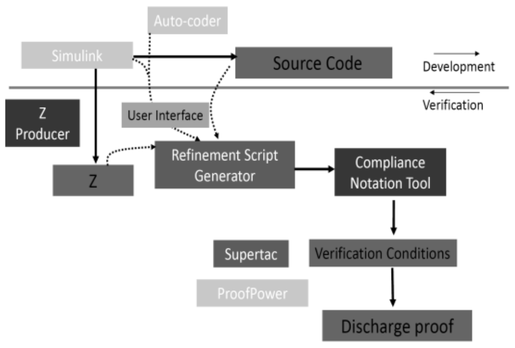
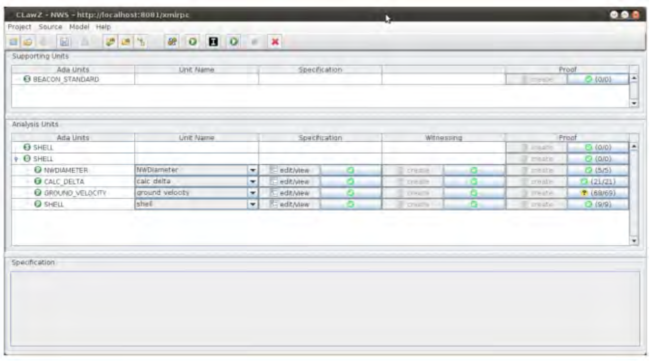


Figure 11

The user interface in Figure 11 is used to set up which code files are to be verified against what Simulink® models. The Z producer automatically generates a specification statement (Morgan 1990), where the pre-condition and post condition are represented in the Z language (Woodcock and Davies 1996). The Refinement Script Generator generates a refinement conjecture, i.e. it constructs a top down refinement as though the code had been incrementally developed from the specification. The compliance tool processes the refinement conjecture and generates verification conditions in the Z language. If the verification conditions are proven then the refinement conjecture holds and the code is a correct implementation of the Simulink® model.

The verification conditions are automatically proven, or simplified for human review. The automated reasoning is carried out by a set of proof tactics called Supertac which direct the theorem prover, called ProofPower (Arthan and Jones 2005), on how to simplify and discharge the verification conditions. The CLawZ toolset was developed and refined over a period of 10 years in order to independently verify the flight and engine control system software as well as the autopilot and auto-throttle software for Eurofighter Typhoon. The semantic validity of the Z producer has been established over that period as well as the effectiveness of the automated proof capability of Supertac.

A screenshot of the user interface for CLawZ after running the velocity estimation software through the verification is shown in Figure 12. All the verification conditions have been proven except for one in the Ada body unit of GROUND_VELOCITY. The simplified verification condition is shown in Figure 13.


CLawZ - NWS - http://localhost:8081/xmlrpc

Project Source Model Help

Supporting Units

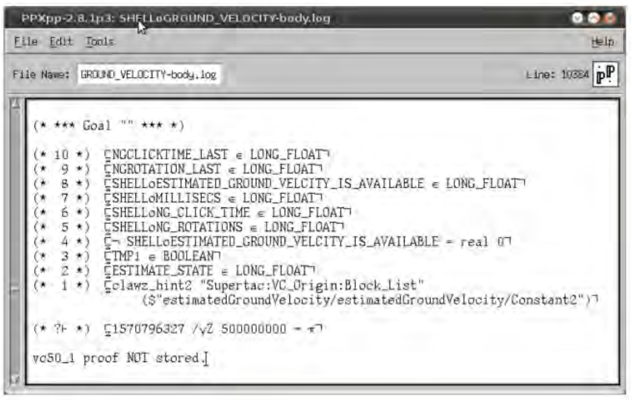
Ada Units	Unit Name	Specification		Proof	
BEACON_STANDARD				create	(0/0)

Analysis Units

Ada Units	Unit Name	Specification		Witnessing		Proof	
SHELL						create	(0/0)
SHELL						create	(0/0)
NWDIAMETER	NWDiameter	edit/view		create		create	(5/5)
CALC_DELTA	calc delta	edit/view		create		create	(21/21)
GROUND_VELOCITY	ground velocity	edit/view		create		create	(68/69)
SHELL	shell	edit/view		create		create	(9/9)

Specification

Figure 12


PPXpp-2.8.1p3: SHELLoGROUND_VELOCITY-body.log

File Edit Tools Help

File Name: GROUND_VELOCITY-body.log Line: 10324

```
(* *** Goal "" *** *)

(* 10 *)  ⌜NGCLICKTIME_LAST ∈ LONG_FLOAT⌝
(*  9 *)  ⌜NGROTATION_LAST ∈ LONG_FLOAT⌝
(*  8 *)  ⌜SHELLoESTIMATED_GROUND_VELCITY_IS_AVAILABLE ∈ LONG_FLOAT⌝
(*  7 *)  ⌜SHELLoMILLISECS ∈ LONG_FLOAT⌝
(*  6 *)  ⌜SHELLoNG_CLICK_TIME ∈ LONG_FLOAT⌝
(*  5 *)  ⌜SHELLoNG_ROTATIONS ∈ LONG_FLOAT⌝
(*  4 *)  ⌜¬ SHELLoESTIMATED_GROUND_VELCITY_IS_AVAILABLE = real 0⌝
(*  3 *)  ⌜TMP1 ∈ BOOLEAN⌝
(*  2 *)  ⌜ESTIMATE_STATE ∈ LONG_FLOAT⌝
(*  1 *)  ⌜clawz_hint2 "Supertac:VC_Origin:Block_List"
              ($"estimatedGroundVelocity/estimatedGroundVelocity/Constant2")⌝

(* ?⊢ *)  ⌜1570796327 /ℤ 500000000 = π⌝

vc50_1 proof NOT stored.
```

Figure 13

The verification condition in Figure 13 is unproven because it is asserting that the transcendental number π is equal to an approximation to π (correct to 9 decimal places). Although the values are unequal, human review would accept the unproven verification condition and pass the code as correct with respect to the algorithm represented by the Simulink® model of Figure 6.

5 Satisfying the Software Requirement

5.1 Formalising the Software Requirement

The previous section has discussed how the code can be formally verified through proof, but it does not show that it satisfies the requirement that *the estimated Ground Velocity shall be within 3 km/hr of the true velocity of the aircraft at some moment within the past 3 seconds*. However it was necessary to develop a Simulink® model to verify the code against. The Simulink® model has a formal mathematical representation and it is this representation in Z that can be used to also prove the software requirement.

Before the software requirement can be proven it must also be expressed formally. A formalisation might take the form of Figure 14.

$$
\begin{array}{l}
\forall t : \mathbb{R} \mid \\
(estimatedGroundVelocityIsAvailable\_at\_time(t + 3)) \\
\bullet \\
\exists t' : \mathbb{R};\ m : \mathbb{N} \bullet \\
((0 <= t') \wedge (t' <= 3)) \wedge ((-3 <= m) \wedge (m <= 3)) \wedge \\
estimatedGroundVelocity\_at\_time\ (t + 3) \\
= \\
trueGroundVelocity\_at\_time\ (t + t') + m
\end{array}
$$

Figure 14

In the Simulink® model of Figure 6 the predicate estimatedGroundVelocityIsAvailable_at_time(t) is a signal that is true if and only if more than 3 seconds has passed and the aircraft is not stationary. The Z that represents this predicate is part of the Z specification generated by the Z producer. Similarly the name estimatedGroundVelocity_at_time(t) is an expression that can be derived from the Z representing the signal estimatedGroundVelocity in Figure 6.

The expression trueGroundVelocity_at_time(t) is a property of the physical model, but it can be conjectured and validated through empirical measurement. Empirical measurement would also be needed to demonstrate the validity of the velocity profile used in the Simulink® simulation. Note that time t is over continuous time not a discrete model of time.

5.2 A Sketch of the Proof

The Simulink® model of Figure 6 is implicitly inside a non-terminating loop, i.e. it is a reactive system. Therefore to prove the formalised requirement of Figure 14 it is necessary to assume both a loop invariant and a predicate that represents the scheduling scheme of the "RTOS model" block of Figure 3. The validity of assuming the loop invariant can be established by standard techniques using the specification for the loop body generated by the Z producer of Figure 11. The predicate for the scheduling scheme that links continuous time with discrete time can be derived from the model within the "RTOS model" block of Figure 3. The proof of the software requirement has not been done and is left as an exercise to the reader.

6 Some Simple Metrics

It takes about 4 minutes to run the automated verification for about 75 lines of non-blank, non-comment code to verify 104 verification conditions for 40 Z paragraphs of specification of mostly a few lines of Z in each paragraph, but a few larger schemas with one of up to 28 conjoined predicates. For larger Simulink® models with an implementation of over a thousand lines it takes a few hours to run the automated verification.

The most striking metrics are that it took 1 day to define the first cut of the Simulink® models. The model was sufficient to validate the algorithm, but it did not fully comply with the subset of Simulink® supported by the CLawZ toolset and it required additional information to enable automatic code generation. A further day was required to:

- modify it so that complied with the CLawZ and auto-coder supported subsets and re-validate;
- automatically generate the Ada using the B4S auto-coder;
- and set up the project through the CLawZ GUI and automatically verify the generated code.

Even if further work had been done to add more details into the Simulink® model and carry out more simulations with validated data, the automated generation and verification would still be as rapid. Any subsequent issues found during rig testing or on-board an aircraft would likely be due to a missing modelling assumption about the physics and hardware aspects of the problem. Even in this case the model could be rapidly changed and verified source code re-generated.

7 Conclusions

The approach advocated in this paper is similar to that put forward by Jones, Hayes and Jackson (2007), but it places it in the context of simulation based modelling of control system engineering. It also takes advantage of model based software development to automate what has previously been seen as a task that required PhD level expertise and cost more than conventional verification through test.

The recently approved guidance for software development DO178-C (RTCA 2012), has a Formal Methods supplement DO-333. The supplement provides guidance on how credit can be taken for the use of Formal Methods and under what circumstances testing can be alleviated. Testing cannot be eliminated, but reductions of at least 60% can be achieved for initial development (O'Halloran 2013). Using DO-333 the CLawZ toolset with qualification material could be used to reduce the cost of testing software. The CLawZ toolset on its own is not sufficient: for example, exception analysis and WCET analysis is required to discharge underlying assumptions.

From the simulation models defined in this paper the following can be obtained:

- a predicate over continuous time
- a scheduling predicate;
- and a predicate over discrete time

can all be derived.

The CLawZ Z producer automatically generates the formal representation of the discrete Simulink® model in Z. ADI's B4S auto-coder independently generates an Ada implementation of the same Simulink® model. CLawZ automatically verifies the functional behaviour of the Ada program against the Z representation by proof.

Satisfaction of a safety requirement of the physical world then becomes a property based upon physics and the mathematical framework enforced by software. This approach allows the reliability of the system to be calculated by established engineering formalisms about physical components that can fail based on the assumption that the software correctly enforces the physical requirements.

8 Acknowledgements

I am grateful to Jeff Joyce for the example of the Nose-Gear challenge and Mike Gordon for inviting me to address the challenge. Andy McCallum for Muretex Ltd. developed the original Simulink® model under my direction. Karen Stephenson prepared the model for automatic code generation and verification using the CLawZ Ada tool. Thanks go to Tony Cant, Jeff Joyce, Clive Boughton, Holger Becht, Brendan Mahony, Rob Arthan and Ursula Martin for reviewing the paper.

9 References

Jones C. B., Hayes I. J., and Jackson M. A. (2007) Deriving specifications for systems that are connected to the physical world. *In Jim Woodcock, editor, Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of their 70th Birthdays,* volume 4700 of Lecture Notes in Computer Science, pages 364–390. Springer Verlag.

Gordon M. (2007) Workshop on theorem proving in certification

http://www.cl.cam.ac.uk/~mjcg/FMStandardsWorkshop/

Joyce, J. (2011) Critical Systems Labs Inc. http://www.cl.cam.ac.uk/~mjcg/FMStandardsWorkshop/NoseGear.html

ADI (2014) http://www.adi.com/products/b4s/

O'Halloran C. (2013) Automated Verification of auto-code from Simulink. *Automated Software Engineering Journal*: Volume 20, Issue 2 (2013), Page 237-264 (DOI) 10.1007/s10515-012-0116-5

Morgan, C. (1990) *Programming from Specifications*. Prentice Hall Series in Computer Science.

Woodcock, J., Davies, J. (1996) *Using Z Specification, Refinement and Proof*, 1st edition. Prentice Hall, New York

Arthan R. D. and Jones R. B. (2005) *Z in HOL in ProofPower*, BCS FACS FACTS

DO-178C (2012), *Software Considerations in Airborne Systems and Equipment Certification* http://www.rtca.org/store_list.asp

Safety Assurance for Operating Procedures – a formal methods approach

Peter A. Lindsay
School of Information Technology and Electrical Engineering,
The University of Queensland
Brisbane, Queensland 4072, Australia

p.lindsay@uq.edu.au

Abstract

Automation is changing the way that many safety critical systems are operated, by changing the nature of the tasks operators perform. In such cases operating procedures need to be redesigned to find a balance between ensuring the operator performs the tasks that the machine requires, while allowing operators the flexibility they require. But how do system designers ensure that human errors are adequately mitigated? Safety standards dictate the degree of rigour that needs to be applied in assurance of software and hardware, but say little about design and verification of operating procedures.

This paper proposes a rigorous qualitative approach to safety evaluation of operating procedures, illustrated on a new Air Traffic Management (ATM) function. Formal models of human-computer and human-human interaction were developed using Behavior Trees, and hazard analysis was performed using automated model checking. Because of the highly interleaved, concurrent nature of the operators' tasks, it was necessary to develop a new way of categorising human error modes and system hazards. Model checking showed that the operating procedures prevented or mitigated errors in many cases, and revealed potential accident sequences in the other cases, thereby effectively validating informal hazard analysis results. The talk will also illustrate use of the safety assurance approach to derive requirements for a modification to the human-computer interface.

Keywords: ATM, formal methods, operating procedures.

1 Introduction

Computers are gradually replacing humans in key roles in safety critical systems (Repperger and Phillips 2009). In many cases the human operator retains the decision-making role but relies increasingly on automation for information and decision options. Yet safety assurance standards are not keeping up. Many of them mandate or highly recommend use of rigorous techniques for specifying, developing and verifying software and hardware, but say little about the design of operating procedures – the human part of the system. For example, IEC 61508 devotes entire parts to assurance techniques for electronics and software (Parts 2 and 3 respectively) but only two pages (Section B.4 in Part 7) to operating procedures, and then only in very general terms (IEC 2010).

1.1 ATM operating procedures

ATM systems are complex federations of many different systems, tools and technologies, typically from different vendors, as well as legacy systems. Often software systems are developed by specialist engineering organisations as "configurable products" and then acquired, configured and integrated by operating organisations, such as Air Navigation Service Providers (ANSPs) in the case of ATM systems. There are standards that offer guidance for the degree of rigour that goes into specification, development and verification of hardware and software – such as DO-278A, an adaptation of the airborne software standard DO-178C for ground-based systems such as ATM (RTCA 2011) – but no equivalent standards for the human element. Yet human error is a prime cause of system failures in complex systems (Reason 1990).

Air Traffic Controllers (ATCos) need to work with multiple systems and tools in parallel. Operation is further constrained by legal requirements (such as Letters of Agreement) governing communications protocols, jurisdiction and handoffs between controllers, protocols for interacting with pilots and other controllers, and so on. Operating procedures need to be designed with all of these factors in mind. For these reasons, it is typically the ANSP who develops, specifies and evaluates the operational procedures rather than the ATM subsystem developers. Human Machine Interfaces (HMIs) and operational procedures need to be designed to reduce the occurrence of errors and to ensure that, if errors occur, they can be detected and corrected easily.

1.2 Use of formal methods

This paper reports on a case study in the use of formal methods in specification and verification of operating procedures for a new mode of working in Air Traffic Management (ATM), based on automated *Trajectory-Based Conflict Detection (TBCD)*. Because of the highly interleaved, concurrent nature of the operator's tasks, it was necessary to develop a new way of categorising human error modes and system hazards, to capture the main concerns of the ATM experts involved.

Formal methods are typically mandated or highly recommended for safety critical systems (Bowen and Stavridou 1993). Formal methods are mathematically based approaches that model the system of interest and analyse its properties and behaviour. They are more rigorous than traditional verification techniques such as review, simulation and testing, and are preferred in principle because they cover all of the state space (that is, all of the conditions which the system may realise) rather than just the parts of the state space that get checked during testing. We contend that this argument applies even more strongly to operating procedures than software, since human-in-the-loop testing is expensive and time-consuming, and almost impossible to configure to test all situations that might arise. Moreover, operator behaviour is highly nondeterministic, with observable differences between individual operators' behaviour, and even differences in the way an individual operator undertakes their task from one situation to the next.

Several different formal methods have been proposed for Human Computer Interaction (HCI) design: Bolton *et al* have written a very comprehensive survey article (Bolton, Bass et al. 2013). For the most part the focus has been on specification and design of the HMI rather than on the operational procedures themselves. Sections 2 and 3 of this paper illustrate the use of a particular formal method – Behavior Trees (Dromey 2006) with automated model checking (Grunske, Lindsay et al. 2005) – on safety evaluation of a collection of proposed operating procedures for a new ATM tool.

In modern safety analysis the term "use error" is often substituted for human error, to capture the idea that some errors are due to poorly designed user interfaces. While this has some relevance to the current study, the issue addressed in this paper is that existing approaches to error categorisation do not apply well, because it is not feasible to define, let alone model, the difference between a correct action and an incorrect action: a wide variety of operator responses are possible, and controllers need the flexibility to apply different tactics. A better approach to evaluation of operating procedures, we contend, is to model system and operator behaviour over time, investigate possible divergences from desired behaviour, check the circumstances under which hazardous states of the system can arise from them, and thereby determine whether operating procedures are adequate to prevent or recover from such divergences. This seems to be similar to Leveson's treatment of accidents as violation of system safety constraints in the STAMP approach (Leveson 2004).

It is not enough however simply to verify safety of a particular design. HCI design is an art: if there are too many operational procedures or they are too complex, the operators won't use them as intended. The HCI designer needs to trade usability off against hazard mitigation. We contend that formal methods can help here, by helping designers and analysts understand how procedures interact, what errors they mitigate and how effective they are as hazard controls, and by supporting derivation of requirements for new HMI features. Section 4 of this paper illustrates this with excerpts from the modelling and analysis we undertook.

2 Background: Trajectory Based Conflict Detection and Resolution

The case study concerns a medium-term conflict detection function, called *Trajectory Based Conflict Detection (TBCD)*. We introduce our own terms here, in part to avoid possible confusion with the evolving concept and in part to harmonise with international terminology.

In short: TBCD operates in airspace where all aircraft are required to report their planned 4D trajectories and seek approval for changes from the assigned controller. TBCD detects possible aircraft conflicts based on the predicted trajectories of flights. Conflict resolution planning and implementation is done by the controller, supported by tools that provide detail about the conflict (such as conflict start time and point of closest approach) and "what if" tools for trialling possible interventions. The operational concept and operational procedures are described in more detail below

2.1 Conflicts and separation standards

To explain the TBCD operational concept further we need to define what is a conflict, which in turn involves explaining separation standards. Loosely speaking, a separation standard is an acceptable "distance" between flights. Distance here might be lateral, longitudinal) or vertical distance, or separation in time. Separation standards often depend on the nature of the airspace (e.g. terminal area vs en route), the equipment on board the aircraft, the nature of surveillance (e.g. radar vs ADSB vs pilot position reports), and more. For our purposes it is enough to know that there are some separation standards that TBCD can monitor and verify by itself, and there are others that require the controller to monitor and verify. They are often simply called procedural separation standards; in what follows the latter are called *Controller Implemented Separation Standards (CISSs)* for clarity.

For our purposes, a conflict occurs between two flights when TBCD detects that, if they continue to follow their current trajectories, at some time in the near future the aircraft will violate all of the separation standards that TBCD can monitor. (In fact there may be other separation standards that TBCD cannot monitor but controllers can, as described below.) Typically the look-ahead time is at least 20 minutes but this can depend on the nature of the airspace and its traffic.

We coin the term *CISS conflict* for a TBCD-detected conflict for which the controller can verify ("establish") that a suitable CISS exists. No intervention is required for CISS conflicts provided the controller continues to monitor the pair and verify that the standard holds. If no CISS can be established then the controller needs to intervene and instruct one or both of the flights to modify their trajectories; for clarity we call these *true conflicts* below. If intervention does not occur early enough, a *Loss of Separation (LoS)* will occur. While there are typically other layers of protection, such as TCAS, LoS is a serious incident for a controller, even if no accident occurs: they will typically be stood down and sent for retraining, or even be dismissed. Note that CISS conflicts need to be continually monitored to detect differences between expected and actual aircraft performance/behaviour; also, an equipment failure or environmental change (such as a GPS RAIM outage) external to the system may cause the CISS to be no longer valid.

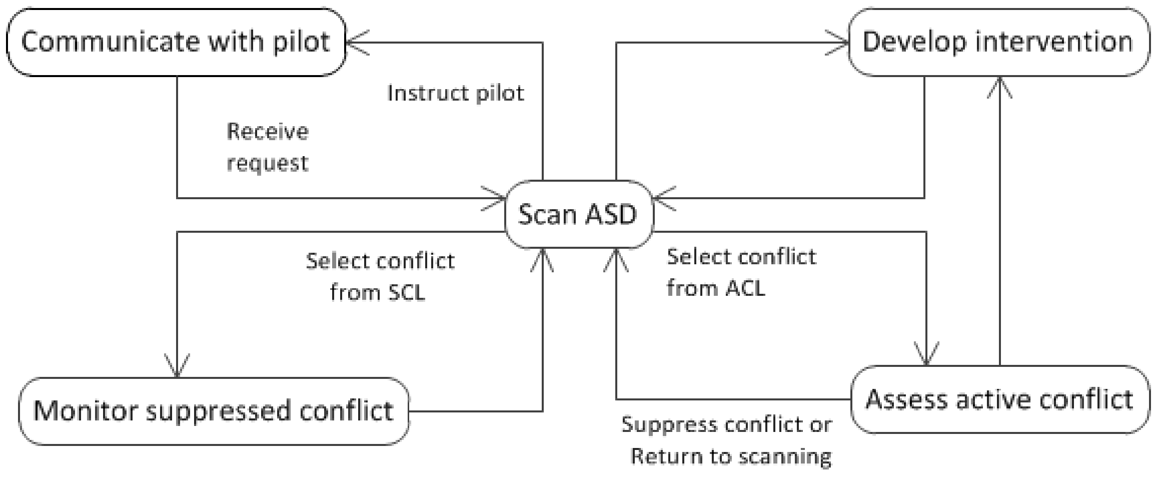


Figure 1. TBCD operator activities

2.2 TBCD operational concept

TBCD accesses the trajectories of aircraft from the ATM system. It maintains two lists of conflicts: the *Active Conflict List (ACL)* and the *Suppressed Conflict List (SCL)*. When a new conflict arises (e.g. because the conflict start time falls within the look-ahead timeframe) it is added to the ACL. The controller can click on a conflict and get more details, to determine whether a CISS applies or whether intervention is required; this process is called *assessment*.

If a CISS can be established, the controller “suppresses” the conflict, which moves it from the ACL to the SCL. If they decide to intervene, they can bring up a what-if tool – called the *Trajectory Modification Tool (TMT)* here – on one of the flights, which allows them to trial a new trajectory for the flight and see what conflicts would result. When they have determined what intervention to apply, they instruct the pilot and click ‘accept’ in TMT, which updates the trajectory in the ATM system. A safety warning alert is activated at a pre-determined interval before conflict start time for conflicts on the ACL; the warnings are suppressed for conflicts on the SCL (as the name suggests).

The main steps are depicted in Figure 1. Note that controllers have other tasks and activities, such as issuing weather reports, ensuring smooth traffic flow, and responding to requests from pilots or other controllers for trajectory modifications. (Typically the controller can only intervene on flights within their sector; otherwise they need to request a change via another controller.) TMT enables them to trial requests before approving them. For the sake of simplicity we’ve only shown interaction with pilots in Figure 1: interaction with other controllers for the purpose of modifying trajectories is analogous. We also don’t distinguish between pilot and co-pilot here, for simplicity.

All these activities are typically carried out in parallel, sometimes with non-trivial delays between one step and the next. This is represented in our model by having the operator return to scanning before undertaking another task.

TCBD has features that can be configured to support operational procedures. Space does not permit a full description of the features here, but for the purposes of this paper we focus on the following features:

- When a new conflict appears on the ACL it is flagged (by displaying it with a coloured border) as being new.
- After the controller has clicked on a new conflict its flag changes to ‘needing assessment’.
- The controller can set a timer on a particular conflict to remind them to come back to it later.

Other ATM functions monitor conformance to trajectories, with procedures for non-compliance.

2.3 Standard Operational Procedures (SOPs)

As noted above, there are many operational procedures for different aspects of the controller’s task. We extracted the procedures that involved use of TBCD functions and developed a Behavior Tree (BT) model from them. Examples include:

1. When a new conflict appears on the ACL, the controller should click on it and view the TBCD display of details of the conflict. (This will result in the new-conflict flag being removed.)
2. After a conflict has been assessed and it has been determined that intervention is required, the controller should clear the ‘needs assessment’ flag. (Thus, any conflict on the ACL without flags implicitly needs intervention.)
3. Before modifying the trajectory of any flight, the controller should open TMT on the flight, enter details of the proposed modification, and check what new conflicts would result. Note that the modification may be acceptable even if conflicts remain – either because the controller may be able to establish a CISS, or because a modification of the other trajectory in the conflict is planned.
4. If the trajectory modification is acceptable, the controller issues the instruction to the pilot and hits ‘accept’ in TMT to update the trajectory in the system.
5. After receiving an instruction the pilot is expected to read it back. The controller should check the read-back instruction against the new trajectory recorded in the system.
6. If the controller determines a CISS exists for a conflict on the ACL, they move the conflict to the

SCL. (The 'needs assessment' flag gets deleted automatically.)

7. The controller should regularly visit each conflict on the SCL to check that the CISS still applies and move it back to the ACL if not.

The natural flow of tasks is either: assess/trial trajectory change/accept and instruct/check read-back, in the case of conflicts requiring intervention; or assess/suppress/monitor in the case of CISS conflicts.

As noted above, each or all of these steps could be interrupted by other tasks, including dealing with other conflicts in parallel, and often can be deferred for significant periods of time. In fact, some conflicts "resolve themselves" (as far as the controller is concerned) because some other agent modifies the trajectory of one of the flights involved, or requests a change (such as a pilot requesting permission to start to descend on approach to its destination). This leads to another SOP:

8. If the controller defers any of the above procedures without completing them, they should set a timer on the conflict to remind them to come back and complete it later. The timer should be set to expire well before conflict start time.

The BT model also captured the behaviour of conflicts (including the fact that a CISS conflict could switch to become a true conflict at any time) and behaviours of pilots (such as responding that they were unable to modify their trajectory as instructed). As noted above, failures of TBCD components were not included in the model as being out of scope, but could have been included if desired.

Figure 2 shows an excerpt from the operator section of the BT model, corresponding to selecting a new conflict c that has appeared on the ACL. (Note that Figure 1 is not BT notation. The BT model has more information than shown in Figure 1, but is too large to display here.) If the controller selects c, a message is sent to the TBCD HMI to remove the new-conflict flag from c, and parameters flight1 and flight2 are set equal to the identities of the two flights involved in c; a "selected" message is returned once the TBCD HMI is updated. The controller can then choose whether to assess c or set a timer and return to scanning. The interested reader is referred to the BT website[1] for details of the syntax and tools available. The "for some" quantifier notation [] c:ACL.new is explained in (Winter, Colvin et al. 2009).

3 Hazard Analysis

3.1 Hazards

As noted above, Loss of Separation (LoS) is the primary TBCD-related safety incident to protect against. But this is already too late: we want to capture states of the system where "trouble is brewing", well before LoS. After discussion with domain experts we arrived at the following states as the system hazards to be analysed:

- *Haz0*: The controller fails to assess a conflict prior to activation of the TBCD safety warning.
- *Haz1*: The TBCD safety warning has been suppressed for a true conflict.
- *Haz2*: The trajectory being flown by the pilot is different from the trajectory in the system.

Haz0 corresponds to a 'late' failure of the conflict detection and resolution (CDR) system function. Haz1 corresponds to an 'omission' failure of the CDR function. Haz2 is a common-mode fault that undermines the integrity of the whole trajectory-based CDR function.

We could have modelled and investigated other system failures – such as unnecessary or inefficient interventions, excess additional controller workload, or pilot requests not being acted on sufficiently early – but they were not as safety critical as the hazards above and so were set aside. (A parallel project evaluated controller workload quantitatively and came up with its own recommendations on workload.) Our methods are concerned with qualitative analysis, as will be explained further below.

3.2 Hazard formalisation

Our analysis method consisted of modelling operator behaviour (as captured in the SOPs) and the behaviour of the environment (in this case, primarily the changeable nature of conflicts and the effects of trajectory modifications on them), then injecting operator failure modes into the model and using automated model checking to determine if any of the hazards became reachable. The interested reader is referred to (Grunske, Lindsay et al. 2005) for more details of the modelling languages and tools involved. The hazards first had to be formalised in *Linear Temporal Logic (LTL),* the logic used by the model checker.

Temporal logic supports reasoning about the orders in which events and conditions may occur. The "linear" part of LTL refers to the fact that every possible sequence of events and conditions in the model get checked. Model checkers have been developed for other forms of temporal logic but they tend to be less efficient, or return less useful results. LTL is good for our purposes because, if the hazard being checked can occur, the model checker returns an example sequence of events (called a *counterexample*) that illustrates how the hazard can occur. Counterexamples are useful for debugging models and, in our case, for identifying possible accident sequences, which in turn reveals where the deficiencies occur in hazard controls.

The first two hazards are straightforward to formalise in LTL. Haz0 can be formulated as "after a new conflict appears, its 'needs assessment' flag should get cleared before the safety warning activates". Haz1 can be formulated simply as the condition that a true conflict appears on the SCL. Haz2 is a little more subtle: "The system trajectory and the pilot trajectory disagree when the controller returns to scanning". We added the qualification "returns to scanning" because instructing the pilot and accepting in TMT occur as separate steps in our model, and hence the hazard would arise every time an intervention occurred without it; this way the steps can occur in either order, but no other steps should be taken in between them.

[1] http://www.itee.uq.edu.au/sse/dccs

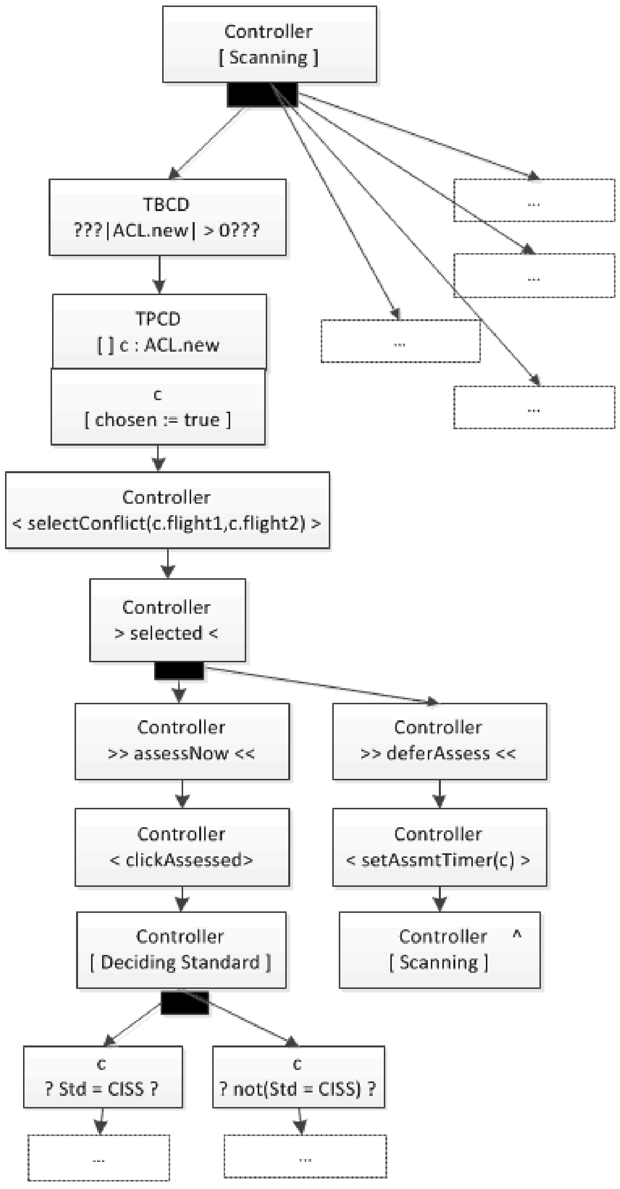


Figure 2. Excerpt from the BT model

Note that temporal logic is concerned only with the order in which things occur, not with when they occur or how long they take to occur: that is, our approach is qualitative rather than quantitative. Methods such as simulation and worst-case real-time analysis can be used for quantitative analysis, but unlike our approach such methods are not exhaustive (in the sense of being able to analyse all of the different cases that might occur). Our models could be used to support quantitative analysis, for example by generating the sequences of events for which the system and pilot trajectories disagree, to help calculate hazard exposure time for Haz2.

3.3 Operator failure modes

The next step in our modelling and analysis was to capture the different kinds of operator error that could occur. (Failures of the TBCD and other ATM system elements were analysed by others and were out of scope of our analysis. The approach could be extended to apply to these factors, as illustrated for example by Grunske et al (Grunske, Winter et al. 2011).)

We generated a variant BT model from the BT model of desired behaviour (i.e., behaviour in accordance with the operating procedures of interest) for each error by the following systematic process: At each point in the BT model where there is an operator action, consider omissions, commissions and incorrect executions. Omission error models are generated by removing one of the actions that are possible at a choice point. Commission error models are generated by adding actions which are not already present but which are possible at that point (i.e., are consistent with the "unconstrained behaviour" from Figure 1). Incorrect-execution error models are generated at parameterised action points (i.e., points where the operator chooses a value for a parameter of a particular action, such as which trajectory to trial in TMT) by replacing the intended value by a different value.

Examples of the errors generated by this process are:

- Inadvertently performing actions. These are similar to Reason's slips and lapses (Reason 1990).
- Performing actions with the wrong (/unintended) data parameter.
- Performing the wrong action – typically here, in contravention of a recommended operating procedure. These are similar to Reason's mistakes.
- Never performing the action that will progress the situation, when some other action (typically deferral) is available.

We claim that, by its systematic nature, this gives complete coverage of possible operator errors. The resulting errors were validated against a list prepared by the ATM experts: where there were differences they were typically matters of detail due to some of the abstractions used in the BT model, or related to timing issues.

It is interesting to compare this approach to other approaches to error categorisation, such as Hollnagel's well-known "error phenotypes" (Hollnagel 1993). His so-called first order error phenotypes included things such as: omission, in the wrong order, wrong action, late and early. But he was dealing primarily with sequential tasks, and many of his phenotypes are difficult to interpret in the current context, where there are multiple (instances of) tasks running in parallel, and task goals and the environment are changing dynamically. To illustrate the difficulties, consider the following vignettes: Controllers often defer certain decisions ("let it run") because the situation will eventually resolve itself; thus for TBCD it is not always possible to say that an omission has occurred, since a task can often be safely deferred. There is no fixed order for doing things: the controller needs flexibility when prioritising and resolving conflicts.

Most HCI formal methods use error type classifications similar in nature to Hollnagel's approach, which has been shown to result in large unwieldy models that are not suited to model checking (Bolton and Bass 2013). We contend that an approach based on functional failure analysis (i.e., where errors are categorised according to how behaviour differs from desired behaviour) is more natural and leads to better insights in-

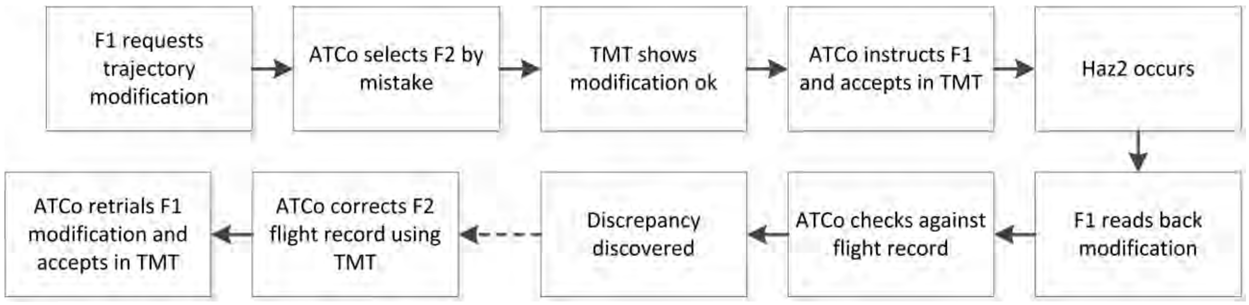

F1 requests trajectory modification
ATCo selects F2 by mistake
TMT shows modification ok
ATCo instructs F1 and accepts in TMT
Haz2 occurs
F1 reads back modification
ATCo checks against flight record
Discrepancy discovered
ATCo corrects F2 flight record using TMT
ATCo retrials F1 modification and accepts in TMT

Figure 3. Accident sequence leading to Haz2

to how to improve operating procedures, as demonstrated below. We think they are also more amenable to model checking, in terms of computational efficiency.

3.4 Model checking

Failure modes were injected into the BT model one by one and the SAL model checker (Moura, Owre et al. 2004) was used to determine if there were any circumstances under which the hazards were reachable. If so, the resulting counterexample was examined to determine which operating procedures were violated (if any) and whether the error would be revealed and recovered from. Recommendations were formulated for additional operating procedures to recover from errors and for modifications to operating procedures to reduce the likelihood of the failure. (The latter was done informally, based on our improved understanding of procedures from modelling.)

Figure 3 illustrates the kind of accident sequence that was revealed by the model checker. In this case the controller has selected the wrong flight in response to a trajectory change request (step 2) and trialled the change on that flight, then approved the original request (step 4). As a result, both flights are flying trajectories different to the ones assigned to them in the ATM system (Haz2). The error is discovered when the pilot of the first flight reads back the instruction and the controller checks it against the system trajectory (step 7). While it is possible they could check it against the other flight's trajectory, the differences should be apparent, so there is low likelihood of a second error compounding the first.

The model checker found the steps through to the hazard occurring and the error being revealed; this shows that the proposed operating procedures are generally adequate for revealing this kind of error. The steps for correcting the error (the last two steps in Figure 3) were added as recommendations for further operating procedures to be developed and stressed in training and assessment. Analysis of the sequence led to discussion about hazard exposure time and a recommendation that policy be developed for how soon the controller should prompt the pilot for read back, if no read back has been received. Consideration of error recovery led to the recommendation that tool support be provided for recovering previous trajectory information; this is discussed further below.

We repeated the automated hazard analysis on variants of the BT model obtained by removing operational procedures one by one. This allowed us to identify which procedures were needed to prevent each hazard – effectively providing traceability from controls to the hazards they prevent

4 Design Improvement

To illustrate the approach's use in design improvement we proposed a modification to the TBCD HMI and repeated the analysis. The proposal consisted of an additional flag on flights, called the *Confirmation Pending Flag (CPF)* here, to indicate whether the trajectory has been changed. The controller would be required to check the details and clear the flag promptly. The CFP should be easy to locate at read-back, and would include details of the original trajectory as well as the change. The TMT should be able to be opened from the original trajectory, if the controller plans to undo the change.

The BT model was changed to incorporate the proposed new feature. The process of integrating the CPF's (and the operator's) behaviour into the model, together with the experience gained in identifying behaviour required for recovering from errors, helped us refine the concept and identify further requirements for CPF. For example, the analysis in Figure 2 above suggested the idea that CPF would be a good place to store the previous trajectory information, in order to easily undo a trajectory change. The idea of accessing TMT from CFP, rather than simply replacing the current trajectory by the old trajectory directly in the ATM system, came from realising that other trajectories may have changed in the meantime, meaning that the controller needs to check that the old trajectory is still safe before rolling back to its old value in the ATM system.

5 Discussion

The Behavior Tree specification language provides an integrated view of the requirements of a system and maintains traceability to the original textual requirements (Dromey 2003). Behavior Trees have been used for modelling large and complex systems and for conducting hazard analyses of such systems (Grunske, Winter et al. 2011, Lindsay, Yatapanage et al. 2012). A key feature of a BT model is its graphical format that makes it easy to understand without a formal methods background. The systematic process of building a BT model, and ease of understanding for non-experts, have been shown to significantly improve the quality of requirements specifications for complex computer-based systems (Powell 2010).

We use fully automated model checking to increase assurance that all cases have been covered. Even with relatively small models the number of different combinations of circumstances (events and component states) that need to be taken into account is difficult for human analysts to handle: automation relieves the tedium and is far less error-prone. The trade-off is in expressiveness of the models and properties to be checked. To avoid the so-called "state explosion problem" of automated model checking we focus on capturing and investigating the "logic" of procedure design and avoid quantitative aspects such as time and physical attributes such as separation distance. Computation time was very reasonable (typically less than 60 seconds per error/hazard combination) but maintaining the failure views was time consuming.

The behaviour model was particularly important for facilitating communications between ATM domain experts and modeller analysts. The modellers would ask probing questions about how the proposed TBCD tools would work, and the modelling notation and model checking forced the team to systematically consider all the different circumstances that could arise. The ATM experts reviewed the models to ensure that functionality and behaviour were captured correctly. One of the main outcomes of the process was a set of recommendations for TBCD operational procedures; the rigour of modelling forced a consistent approach across the system and increased confidence that hazard analysis was applied across the TBCD concept completely. Overall the process significantly increased the quality of safety requirements specification and assurance.

The BT notation has no way of specifying behaviours such as the requirement that the operator should regularly check for flagged items and deal with them promptly. It was necessary to reformulate the safety properties to be model checked so as to include "fairness conditions" for such behaviours. This part of the analysis was tricky; we are investigating how best to model and check such procedures.

As noted above, the approach is not suited to quantitative analysis: techniques such as Event Tree Analysis (Storey 1996) or THERP (Kirwan, Gibson et al. 2008) would need to be applied to determine whether HMI features and requirements such as the above really do reduce risk effectively.

6 Related work

BT support for Functional Failure Analysis and FMEA has been investigated in a number of domains, including manufacturing and medical devices (Grunske, Winter et al. 2011) and aerial firefighting (Lindsay, Winter et al. 2012). The approach has been adapted to Cut Set Analysis (discovering combination of component failures that can lead to hazards) in aerospace (Lindsay, Yatapanage et al. 2012).

There is a wide range of HCI safety analysis techniques for socio-technical systems, including ATM. Leveson et al (Leveson, de Villepin et al. 2001) propose a human-centred, safety-driven design process with wide-ranging coverage; our approach focuses on modelling operational concepts and using model checking to automate analysis, and yield more objective and repeatable results. The HERA project (Isaac, Shorrock et al. 2002) analysed ATM human error from a cognitive viewpoint: our approach to error identification was informed by their approach. Most human performance modelling approaches (e.g. (Blom, Daams et al. 2000)) evaluate risk on a scenario basis, whereas our approach applies directly to the operational concept as a whole.

Formal methods have been applied to HCI for a wide variety of purposes: see Bolton et al (Bolton, Bass et al. 2013) for an excellent review. Paterno et al (Paternò and Santoro 2001) apply model checking to ConcurTaskTrees (CTT) models translated to LOTOS, to verify user interface protocols, using a case study from ATM. In (Paterno and Santoro 2002) they show how CTT can be used to evaluate HCI design options from safety and usability viewpoint; it is not clear how to apply the approach post hoc to an existing HMI.

Bolton et al (Bolton, Siminiceanu et al. 2011) introduce the Enhanced Operator Function Model (EOFM) notation and apply the SAL model checker to find a failure pathway, via a counterexample, for an automotive cruise-control case study. They then propose a design change and show that the hazard is no longer reachable. Like BTs, EOFM is a graphical notation. EOFM has the advantage over BTs that models can be represented hierarchically; the downside is the state explosion problem seems to occur earlier for hierarchical models due to the extra overhead. It is not clear how well EOFM would cope with a task model as complex as TBCD nor how easily failure modes could be injected into the model for capturing human errors. In (Bolton, Bass et al. 2012) they propose a way of automating generation of failure models but quickly run into the state explosion problem.

7 Conclusion

The paper describes the application of formal methods to modelling and safety analysis of operating procedures for a Trajectory-Based Conflict Detection (TBCD) function for airspace that includes procedural separation standards. The paper evaluates a set of HMI features and operating procedures designed as hazard controls, to maintain system safety and integrity. The robustness of the procedures is evaluated in terms of which operator errors and system hazards they mitigate. It introduces a new approach to classification and analysis of operator errors for highly interleaved concurrent tasks, addressing a limitation of existing approaches by enabling operator actions to be judged dynamically over time and in a systems context.

The results in this paper were derived using the Behavior Tree modelling notation and the SAL symbolic model checker. Formal modelling expedited consideration of cross-system interactions and dependencies and enforced consistency across the design. It also provided an objective basis for error categorisation. Model checking automated the consequence analysis, relieving the analyst of one of the most labour-intensive and error-prone aspects of analysis, and increased repeatability of the analysis and assurance that all cases have been taken into account. Overall the process significantly increased the quality of (qualitative) safety requirements specification and assurance.

Acknowledgements
The author gratefully acknowledges the assistance of Nisansala Yatapanage and Soon Kyeong Kim in development of the formal models and model checking, and Andrew Neal, Ed Macfarlane, Damien Armenis and Phil Hong for discussions and review in the ATM domain.

8 References

Blom, H. A., J. Daams and H. B. Nijhuis (2000). Human cognition modelling in ATM safety assessment. 3rd USA/Europe Air Traffic Management R&D Seminar, NLR.

Bolton, M. L. and E. J. Bass (2013). "Generating erroneous human behavior from strategic knowledge in task models and evaluating its impact on system safety with model checking." Systems, Man, and Cybernetics: Systems, IEEE Transactions on **43**(6): 1314-1327.

Bolton, M. L., E. J. Bass and R. I. Siminiceanu (2012). "Generating phenotypical erroneous human behavior to evaluate human–automation interaction using model checking." International Journal of Human-Computer Studies **70**(11): 888-906.

Bolton, M. L., E. J. Bass and R. I. Siminiceanu (2013). "Using formal verification to evaluate human-automation interaction: a review." Systems, Man, and Cybernetics: Systems, IEEE Transactions on **43**(3): 488-503.

Bolton, M. L., R. I. Siminiceanu and E. J. Bass (2011). "A systematic approach to model checking human-automation interaction using task analytic models." Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **41**(5): 961-976.

Bowen, J. and V. Stavridou (1993). "Safety-critical systems, formal methods and standards." Software Engineering Journal **8**(4): 189-209.

Dromey, R. G. (2003). From requirements to design: Formalizing the key steps. Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on, IEEE**:** 2-11.

Dromey, R. G. (2006). "Climbing over the "No Silver Bullet" Brick Wall." IEEE Software **23**(2): 120-119.

Grunske, L., P. Lindsay and K. Winter (2005). An automated failure mode and effect analysis based on high-level design specification with Behavior Trees. Proceedings Integrated Formal Methods. Berlin, Heidelberg, Springer. **3771:** 129-149.

Grunske, L., K. Winter, N. Yatapanage, S. Zafar and P. A. Lindsay (2011). "Experience with fault injection experiments for FMEA." Software: Practice and Experience **41**(11): 1233-1258.

Hollnagel, E. (1993). "The phenotype of erroneous actions." International Journal of Man-Machine Studies **39**(1): 1-32.

IEC (2010). 61508: Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety-related Systems, International Electrotechnical Commission.

Isaac, A., S. T. Shorrock and B. Kirwan (2002). "Human error in European air traffic management: the HERA project." Reliability Engineering & System Safety **75**(2): 257-272.

Kirwan, B., W. H. Gibson and B. Hickling (2008). "Human error data collection as a precursor to the development of a human reliability assessment capability in air traffic management." Reliability Engineering & System Safety **93**(2): 217-233.

Leveson, N., M. de Villepin, J. Srinivasan, M. Daouk, N. Neogi, E. Bachelder, J. Bellingham, N. Pilon and G. Flynn (2001). A safety and human-centered approach to developing new air traffic management tools. Proceedings Fourth USA/Europe Air Traffic Management R&D Seminar**:** 1-14.

Leveson, N. G. (2004). "A systems-theoretic approach to safety in software-intensive systems." IEEE Transactions on Dependable and Secure Computing **1**(1): 66-86.

Lindsay, P. A., K. Winter and S. Kromodimoeljo (2012). Model-based safety risk assessment using Behavior Trees. Asia Pacific Conference on Systems Engineering (APCOSE)/Australian Systems Engineering, Test & Evaluation (SETE) 2012 combined conference, Systems Engineering Society of Australia.

Lindsay, P. A., N. Yatapanage and K. Winter (2012). "Cut Set Analysis using Behavior Trees and model checking." Formal Aspects of Computing **24**(2): 249-266.

Moura, L., S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea and A. Tiwari (2004). SAL 2. Computer Aided Verification. R. Alur and D. Peled, Springer Berlin Heidelberg. **3114:** 496-500.

Paterno, F. and C. Santoro (2002). "Preventing user errors by systematic analysis of deviations from the system task model." International Journal of Human-Computer Studies **56**(2): 225-225.

Paternò, F. and C. Santoro (2001). Integrating model checking and HCI tools to help designers verify user interface properties. Interactive Systems Design, Specification, and Verification, Springer**:** 135-150.

Powell, D. (2010). Behavior engineering-a scalable modeling and analysis method. Software Engineering and Formal Methods (SEFM), 2010 8th IEEE International Conference on, IEEE**:** 31-40.

Reason, J. T. (1990). Human error. Cambridge, England ; New York :, Cambridge University Press.

Repperger, D. W. and C. A. Phillips (2009). The Human Role in Automation. Springer Handbook of Automation, Springer**:** 295-304.

RTCA (2011). DO-278A: Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems (aka EUROCAE ED-109A), Radio Technical Commission for Aeronautics.

Storey, N. R. (1996). Safety Critical Computer Systems, Addison-Wesley Longman Publishing Co., Inc.

Winter, K., R. Colvin and R. G. Dromey (2009). Dynamic relational behaviour for large-scale systems. Australian Software Engineering Conference (ASWEC)**:** 173-182.

Domain Engineering
A Basis for Safety Critical Software

Dines Bjørner

Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~dibj

Abstract

Before software can be designed we must have a reasonable grasp of the requirements that the software is supposed to fulfil. And before requirements can be prescribed we must have a reasonable grasp of the "underlying" application domain. Domain engineering now becomes a software engineering development phase in which a precise description, desirably formal, of the domain within which the target software is to be embedded. Requirements engineering then becomes a phase of software engineering in which one systematically derives requirements prescriptions from the domain description. (Software design is then the software engineering phase which (also) results in code.) We illustrate the first element, $\mathcal{D}$, of this triptych ($\mathcal{D}, \mathcal{R}, \mathcal{S}$) by an example, Sect. 2, in which we show a description of a pipeline domain where, for example, the operations of pumps and valves are safety critical. We then, Sects. 3–5, summarise the methodological stages and steps of domain engineering. We finally weave considerations of *domain safety criticality*[1] into a section (Sect. 5) on domain facets. We believe this aspect of safety criticality is new: the connection of issues of safety criticality to domain engineering. The study presented here need be deepened. Similar connections need be made to requirements engineering such as it can be "derived" from domain engineering (Bjørner 2008), and to the related software design. That is, three distinct "layers" of safety engineering.

1 Introduction

1.1 A Software Development Triptych

Before software can be designed we must have a reasonable grasp of the requirements that the software is supposed to fulfil. And before requirements can be prescribed we must have a reasonable grasp of the "underlying" application domain. **Domain engineering** now becomes a software engineering development phase in which a precise description, desirably formal, of the domain within which the target software is to be embedded. **Requirements engineering** then becomes a phase of software engineering in which one systematically derives requirements prescriptions from the domain description — carving out and extending, as it were, a subset of those domain properties that are computable and for which computing support is required. **Software design** is then the software engineering phase which results in code (and further documentation). .

1.2 Domain Description

To us a domain description is a set of pairs of narrative, that is, informal, and formal description texts. The narrative texts should go hand-in-hand with the formal texts; that is, the narrative should be "a reading" of the formalisation; and the formalisation should be an abstraction that emphasises properties of the domain, not some intricate, for example, "executable" model of the domain.[2] These "pairings" will be amply illustrated in Sect. 2. The meaning of a domain description is basically a heterogeneous algebra[3], that is: sets of typed entities and a set of typed operations over these. The formalisation language is here the `RAISE` (George et al. 1995) Specification Language `RSL` (George et al. 1992); but it could be any of, for example, `Alloy` (Jackson 2006), `Event B` (Abrial 2009*a*), `VDM-SL` (Bjørner & Jones 1978, 1982, Fitzgerald & Larsen 1998) or `Z` (Woodcock & Davies 1996). That is, the main structure of the description of the domain endurants, such as we shall advocate it, may, by some readers, be thought of as an ontology (Benjamin & Fensel 1998, Fox 2000). But our concept a domain description is a much wider concept of ontology than covered by (Benjamin & Fensel 1998); it is more in line with (Mellor & Oliver 1997, Fox 2000).

1.3 A Domain Description "Ontology"

We shall, in Sect. 2, give a fairly large example, approximately 3.5 Pages, of a postulated domain of (say, oil or gas) pipelines; the focus will be on **endurant**s: the observable **entities** that endure, their **mereology**, that is, how they relate, and their **attribute**s. **Perdurant**s: **action**s, **event**s and **behaviour**s will be very briefly mentioned.

We shall then, in Sect. 3 on the background of this substantial example, outline the basic principles, techniques and tools for describing domains — focusing only on endurants.

The mathematical structure that is built up when describing a domain hinges on the following elements: there are *entities;* entities are either *endurants* or *perdurants;* endurants are either *discrete* or *continuous;* discrete endurants are also called *parts;* continuous endurants are also called *materials;* parts are either

[1]– as opposed to 'system safety criticality'

[2]Domain descriptions are usually not deiscriptions of computable phenomena.

[3]This is just one of the ways in which a domain description differs from an ontology.

atomic or *composite;* parts have *unique identifiers, mereologies* and *attributes;* materials have attributes; so entities are what we see and unique identifiers, mereologies and attributes are entity qualities. A domain description is then composed from one or more part and material descriptions; descriptions of unique part identifiers, part mereologies and part attributes. This structure that, to some, may remind them of an *"upper ontology."* Different domain descriptions all basically have the same "upper ontology."

1.4 A Method: its Principles, Tools and Techniques

By a **method** we shall understand a set of **principles** of **selecting** and **applying** a number of **techniques** and **tools,** for **analysing** and **constructing** an artefact. By a **formal method** we shall understand a set of a method whose techniques and tools can be given a mathematical basis that enable formal reasoning.

The **principles** of our approach to domain analysis and description are embodies in the above "upper ontology". The **tools of analysis** are embodied in a number of **domain analysis prompts**. Analysis prompts form a comprehensive and small set of predicates mentally "executed" by the domain analyser. The **tools of description** are embodied in a number of **domain description prompts**. Description prompts form a comprehensive and small set of RSL-text generating functions mentally "executed" by the domain describer. The domain analyser and describer is usually one and the same person the domain engineer cum scientist. The analysis and description **techniques** are outlined in the texts of Sects. 3 and 5. We claim that this formulation of the concept of method and formal method, their endowment with prompt tools, and the underlying techniques is new.

1.5 Safety Criticality

In Sect. 4 we shall review notions of **safety criticality**: **safety**, **failure**, **error**, **fault**, **hazard** and **risk**. We emphasize that we are focusing sôlely on issues of **domain safety**. That is, we are not dealing with **system safety** where we understand the term 'system' to designate a composition of software and hardware that is being designed in order to solve problems, including such which may aris from issues of domain safety. Finally, in Sect. 5, we shall detail the notion of **domain facet**s. The various domain facets somehow reflect domain views — of logical or algebraic nature — views that are shared across stake-holder groups, but are otherwise clearly separable. It is in connection with the summary explanation of respective domain facets that we identify respective **faults** and **hazards.** The presentation is brief. We refer to (Bjørner 2010*a*) for a more thorough coverage of the notion of domain facets.

1.6 Contribution

We consider the following ideas new: the idea of describing domains before prescribing requirements (but see (Bjørner 2006*c*, Part IV, 2006), (Bjørner 2007, 2007), (Bjørner 2010*a*, written in 2007, published in 2010), (Bjørner 2008, 2008), (Bjørner 2010*b*, 2011*a*, 2010), and (Bjørner 2014*b*, 2014)), and the idea of enumerating faults and hazards as related to individual facets. For the latter "discovery" we thank the organisers of ASSC 2014, notably Prof. Clive Victor Boughton.

2 An Example

Our example is an abstraction of pipeline system endurants. The presentation of the example reflects a rigorous use of the domain analysis & description method outlined in Sect. 3, but is relaxed with respect to not showing all – one could say intermediate – analysis steps and description texts, but following stoichiometry ideas from chemistry makes a few short-cuts here and there. The use of the "stoichiometrical" reductions, usually skipping intermediate endurant sorts, ought properly be justified in each step — and such is adviced in proper, industry-scale analyses & descriptions.

To guide your intuition with respect to what a pipeline system might be we suggest some diagrams and some pictures. See Figs. 1 and 2.

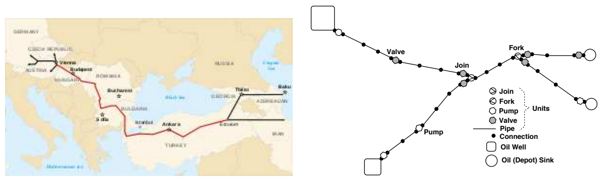


Figure 1: Pipelines. Flow is right-to-left in left figure, but left-to-right in right figure.

The description only covers a few aspects of endurants.

Figure 2: Pump, pipe and valve pipeline units

2.1 Parts

1. A pipeline system contains a set of pipeline units and a pipeline system monitor.
2. The well-formedness of a pipeline system depends on its mereology (cf. Sect. 2.2.3) and the routing of its pipes (cf. Sect. 2.3.2).
3. A pipeline unit is either a well, a pipe, a pump, a valve, a fork, a join, or a sink unit.
4. We consider all these units to be distinguishable, i.e., the set of wells, the set pipe, etc., the set of sinks, to be disjoint.

```
    type
1.    PLS′, U, M[4]
2.    PLS = {| pls:PLS′•wf_PLS(pls) |}[5]
    value
2.    wf_PLS: PLS′ → Bool[6]
2.    wf_PLS(pls) ≡
2.       wf_Mereology(pls) ∧ wf_Routes(pls)[7]
1.    obs_Us: PLS → U-set[8]
1.    obs_M: PLS → M[9]
    type
3.    U = We | Pi | Pu | Va | Fo | Jo | Si[10]
4.    We :: Well[11]
```

```
4.    Pi :: Pipe
4.    Pu :: Pump
4.    Va :: Valv
4.    Fo :: Fork
4.    Jo :: Join
4.    Si :: Sink
```

2.2 Part Identification and Mereology

2.2.1 Unique Identification.

5. Each pipeline unit is uniquely distinguished by its unique unit identifier.

```
    type
5.     UI
    value
5.     uid_UI: U → UI[12]
    axiom
5.     ∀ pls:PLS,u,u′:U•
5.         {u,u′}⊆obs_Us(pls)⇒
5.             u≠u′⇒uid_UI(u)≠uid_UI(u′)[13]
```

2.2.2 Unique Identifiers.

6. From a pipeline system one can observe the set of all unique unit identifiers.

```
    value
6.  xtr_UIs: PLS → UI-set[14]
6.  xtr_UIs(pls) ≡ {uid_UI(u)|u:U•u ∈ obs_Us(pls)}
```

7. We can prove that the number of unique unit identifiers of a pipeline system equals that of the units of that system.

```
    theorem:
7.     ∀ pls:PLS•card obs_Us(pl)=card xtr_UIs(pls)
```

2.2.3 Mereology.

8. Each unit is connected to zero, one or two other existing (formula line 8x.) input units and zero, one or two other existing (formula line 8x.) output units as follows:
    a A well unit is connected to exactly one output unit (and, hence, has no "input").
    b A pipe unit is connected to exactly one input unit and one output unit.
    c A pump unit is connected to exactly one input unit and one output unit.
    d A valve is connected to exactly one input unit and one output unit.
    e A fork is connected to exactly one input unit and two distinct output units.
    f A join is connected to exactly two distinct input units and one output unit.
    g A sink is connected to exactly one input unit (and, hence, has no "output").

```
    type
8.     MER = UI-set × UI-set
    value
8.     mereo_U: U → MER
    axiom
8.     wf_Mereology: PLS → Bool
8.     wf_Mereology(pls) ≡
8.        ∀ u:U•u ∈ obs_Us(pls)⇒
8x.            let (iuis,ouis) = mereo_U(u)[15] in
8x.            iuis ∪ ouis ⊆ xtr_UIs(pls)[16]∧
8.               case (u,(card iuis,card ouis)) of[17]
8a                     (mk_We(we),(0,1)) → true,[18]
8b                     (mk_Pi(pi),(1,1)) → true,[19]
8c                     (mk_Pu(pu),(1,1)) → true,
8d                     (mk_Va(va),(1,1)) → true,
8e                     (mk_Fo(fo),(1,2)) → true,[20]
8f                     (mk_Jo(jo),(2,1)) → true,[21]
8g                     (mk_Si(si),(1,0)) → true,
8.                     _ → false end end
```

2.3 Part Concepts

An aspect of domain analysis & description that was not covered in Sects. 2.1–2.2 was that of derived concepts. Example pipeline concepts are routes, acyclic or cyclic, circular, etcetera. In expressing well-formedness of pipeline systems one often has to develop subsidiary concepts such as these by means of which well-formedness is then expressed.

2.3.1 Pipe Routes.

9. A route (of a pipeline system) is a sequence of connected units (of the pipeline system).

10. A route descriptor is a sequence of unit identifiers and the connected units of a route (of a pipeline system).

```
    type
9.     R′ = U^ω[23]
9.     R = {| r:Route′•wf_Route(r) |}
10.    RD = UI^ω
    axiom
10.    ∀ rd:RD • ∃ r:R•rd=descriptor(r)
    value
10.    descriptor: R → RD[24]
10.    descriptor(r) ≡ ⟨uid_UI(r[ i ])|i:Nat•1≤i≤len r⟩
```

[4]PLS′, US, U and M are being defined as sorts, i.e., sets of endurant entities.

[5]PLS is the subtype (i.e., subset) of well-formed PLS entities.

[6]wf_PLS is the PLS well-formedness predicate whose signature (i.e., type) is that of a function from PLS′ entities to truth values in **Bool**.

[7]wf_PLS(pls) is defined to be the conjunction of the well-formedness of the mereology of pls and pls defining only well-formed routes.

[8]obs_US is an observer function which maps plss into sets of units.

[9]obs_M is an observer function which maps plss into a monitor.

[10]U is defined to be the discriminated (::) union (|) of sorts We, Pi, Pu, Va, Fo, Jo and Si.

[11]We is discriminated from Pi, Pu, Va, Fo, Jo and Si by the constructor: :: mkWell, etcetera.

[12]uid_UI is the unique identifier observer function for parts u:U. It is total. uid_UI(u) yields the unique identifier of u.

[13]The axiom expresses that for all pipeline systems all two distinct units, u, u′ of such pipeline systems have distinct unique identifiers.

[14]xtr_UIs is a total function. It extracts all unique unit identifiers of a pipeline system.

[15]The **let** clause names the pair resulting from mereo_U(u).

[16]The input and out unique identifiers are a subset of all pipe line unit unique identifiers.

[17]This **case**..**pattern**..**end** clause "sequentially matches" the pattern "against" the →.. clauses.

[18]Wells have 0 input and 1 output.

[19]Pipes, Pumps and Valves have 1 input and 1 out.

[20]Forks have 1 input and 2 outputs.

[21]Joins have 2 input and 1 output.

[22]Sinks have 1 input and 0 output.

11. Two units are adjacent if the output unit identifiers of one shares a unique unit identifier with the input identifiers of the other.

```
    value
11.     adjacent: U × U → Bool
11.     adjacent(u,u′) ≡
11.         let (,ouis)=mereo_U(u),(iuis,)=mereo_U(u′) in
11.         ouis ∩ iuis ≠ {} end
```

12. Given a pipeline system, *pls*, one can identify the (possibly infinite) set of (possibly infinite) routes of that pipeline system.
    a The empty sequence, $\langle\rangle$, is a route of *pls*.
    b Let u, u' be any units of *pls*, such that an output unit identifier of u is the same as an input unit identifier of u' then $\langle u, u'\rangle$ is a route of *pls*.
    c If r and r' are routes of *pls* such that the last element of r is the same as the first element of r', then r^**tl**r' is a route of *pls*.
    d No sequence of units is a route unless it follows from a finite (or an infinite) number of applications of the basis and induction clauses of Items 12a–12c.

```
     value
12.  Routes: PLS → RD-infset[25]
12.  Routes(pls) ≡
12a.     let rs = ⟨⟩ ∪
12b.             {⟨uid_UI(u),uid_UI(u′)⟩|u,u′:U•{u,u′}
12b.             ⊆ obs_Us(pls) ∧ adjacent(u,u′)}
12c.          ∪ {r^tl r′|r,r′:R•{r,r′}⊆rs}[26]
12d.     in rs[27] end
```

2.3.2 Well-formed Routes.

13. A route is acyclic if no two route positions reveal the same unique unit identifier.

```
    value
13. acyclic_Route: R → Bool
13. acyclic_Route(r) ≡
13.     ∼∃ i,j:Nat•{i,j}⊆inds r ∧ i≠j ∧ r[i]=r[j]
```

14. A pipeline system is well-formed if none of its routes are circular (and all of its routes embedded in well-to-sink routes).

```
    value
14. wf_Routes: PLS → Bool
14. wf_Routes(pls) ≡
14.     non_circular(pls) ∧
14.     embedded_in_well_to_sink_Routes(pls)

14. non_circular_PLS: PLS → Bool
14. non_circular_PLS(pls) ≡
14.     ∀ r:R•r ∈ routes(p)∧acyclic_Route(r)
```

[23] U^{ω} denotes the class of finite and infinite length sequences of U elements.

[24] The descriptor function converts a finite or infinite length sequence of U elements to a "corresponding length" UI elements.

[25] The Routes function generates the potentially infinite set of routes of a pipe line system.

[26] The **let** rs = ... clause is defined recursively and (cf. Footnote 27).

[27] rs is the smallest set which satisfies the **let** rs = ... equation..

15. We define well-formedness in terms of well-to-sink routes, i.e., routes which start with a well unit and end with a sink unit.

```
    value
15. well_to_sink_Routes: PLS → R-set
15. well_to_sink_Routes(pls) ≡
15.     let rs = Routes(pls) in
15.     {r|r:R•r ∈ rs ∧
15.     is_We(r[1]) ∧ is_Si(r[len r])} end
```

16. A pipeline system is well-formed if all of its routes are embedded in well-to-sink routes.

```
16. embedded_in_well_to_sink_Routes: PLS → Bool
16. embedded_in_well_to_sink_Routes(pls) ≡
16.     let wsrs = well_to_sink_Routes(pls) in
16.     ∀ r:R • r ∈ Routes(pls) ⇒
16.         ∃ r′:R,i,j:Nat •
16.             r′ ∈ wsrs
16.           ∧ {i,j}⊆inds r′∧i≤j
16.           ∧ r = ⟨r′[k]|k:Nat•i≤k≤j⟩ end
```

2.3.3 Embedded Routes.

17. For every route we can define the set of all its embedded routes.

```
    value
17. embedded_Routes: R → R-set
17. embedded_Routes(r) ≡
17.     {⟨r[k]|k:Nat•i≤k≤j⟩
17.         | i,j:Nat• i {i,j}⊆inds(r) ∧ i≤j}
```

2.3.4 A Theorem.

18. The following theorem is conjectured:
    a the set of all routes (of the pipeline system)
    b is the set of all well-to-sink routes (of a pipeline system) and
    c all their embedded routes

```
    theorem:
18.  ∀ pls:PLS •
18.  let rs = Routes(pls),
18.      wsrs = well_to_sink_Routes(pls) in
18a. rs =
18b.     wsrs ∪
18c.     ∪ {{r′|r′:R • r′ ∈ embedded_Routes(r″)}
18c.             | r″:R • r″ ∈ wsrs}
17.  end
```

2.4 Materials

19. The only material of concern to pipelines is the gas[28] or liquid[29] which the pipes transport[30].

```
    type
19.     GoL
    value
19.     obs_GoL: U → GoL
```

[28] Gaseous materials include: air, gas, etc.

[29] Liquid materials include water, oil, etc.

[30] The description of this paper is relevant only to gas or oil pipelines.

2.5 Attributes

2.5.1 Part Attributes.

20. These are some attribute types:
    - a estimated current well capacity (barrels of oil, etc.),
    - b pipe length,
    - c current pump height,
    - d current valve open/close status and
    - e flow (e.g., volume/second).

```
    type
20a.   WellCap
20b.   LEN
20c.   Height
20d.   ValSta == open | close
20e.   Flow
```

21. Flows can be added (also distributively) and subtracted, and
22. flows can be compared.

```
    value
21.    ⊕,⊖: Flow×Flow → Flow
21.    ⊕: Flow-set → Flow
22.    <,≤,=,≠,≥,>: Flow × Flow → Bool
```

23. Properties of pipeline units include
    - a estimated current well capacity (barrels of oil, etc.),
    - b pipe length,
    - c current pump height,
    - d current valve open/close status,
    - e current $\mathcal{L}$aminar in-flow at unit input,
    - f current $\mathcal{L}$aminar in-flow leak at unit input,
    - g maximum $\mathcal{L}$aminar guaranteed in-flow leak at unit input,
    - h current $\mathcal{L}$aminar leak unit interior,
    - i current $\mathcal{L}$aminar flow in unit interior,
    - j maximum $\mathcal{L}$aminar guaranteed flow in unit interior,
    - k current $\mathcal{L}$aminar out-flow at unit output,
    - l current $\mathcal{L}$aminar out-flow leak at unit output,
    - m maximum guaranteed $\mathcal{L}$aminar out-flow leak at unit output.

```
    value
23a.   attr_WellCap: We → WellCap
23b.   attr_LEN: Pi → LEN
23c.   attr_Height: Pu → Height
23d.   attr_ValSta: Va → VaSta
23e.   attr_In_Flow_$\mathcal{L}$: U → UI → Flow
23f.   attr_In_Leak_$\mathcal{L}$: U → UI → Flow
23g.   attr_Max_In_Leak_$\mathcal{L}$: U → UI → Flow
23h.   attr_body_Flow_$\mathcal{L}$: U → Flow
23i.   attr_body_Leak_$\mathcal{L}$: U → Flow
23j.   attr_Max_Flow_$\mathcal{L}$: U → Flow
23k.   attr_Out_Flow_$\mathcal{L}$: U → UI → Flow
23l.   attr_Out_Leak_$\mathcal{L}$: U → UI → Flow
23m.   attr_Max_Out_Leak_$\mathcal{L}$: U → UI → Flow
```

2.5.2 Flow Laws.

24. "What flows in, flows out !". For $\mathcal{L}$aminar flows: for any non-well and non-sink unit the sums of input leaks and in-flows equals the sums of unit and output leaks and out-flows.

```
    Law:
24.    ∀ u:U\We\Si •
24.       sum_in_leaks(u) ⊕ sum_in_flows(u) =
24.       attr_body_Leak_$\mathcal{L}$(u) ⊕
24.       sum_out_leaks(u) ⊕ sum_out_flows(u)
```

```
value
   sum_in_leaks: U → Flow
   sum_in_leaks(u) ≡
      let (iuis,) = mereo_U(u) in
      ⊕ {attr_In_Leak_$\mathcal{L}$(u)(ui)|ui:UI•ui ∈ iuis} end
   sum_in_flows: U → Flow
   sum_in_flows(u) ≡
      let (iuis,) = mereo_U(u) in
      ⊕ {attr_In_Flow_$\mathcal{L}$(u)(ui)|ui:UI•ui ∈ iuis} end
   sum_out_leaks: U → Flow
   sum_out_leaks(u) ≡
      let (,ouis) = mereo_U(u) in
      ⊕ {attr_Out_Leak_$\mathcal{L}$(u)(ui)|ui:UI•ui ∈ ouis} end
   sum_out_flows: U → Flow
   sum_out_flows(u) ≡
      let (,ouis) = mereo_U(u) in
      ⊕ {attr_Out_Leak_$\mathcal{L}$(u)(ui)|ui:UI•ui ∈ ouis} end
```

25. "What flows out, flows in !". For $\mathcal{L}$aminar flows: for any adjacent pairs of units the output flow at one unit connection equals the sum of adjacent unit leak and in-flow at that connection.

```
    Law:
25. ∀ u,u′:U•adjacent(u,u′) ⇒
25.   let (,ouis) = mereo_U(u),
25.       (iuis′,) = mereo_U(u′) in
25.   uid_U(u′) ∈ ouis ∧ uid_U(u) ∈ iuis′ ∧
25.   attr_Out_Flow_$\mathcal{L}$(u)(uid_U(u′)) =
25.   attr_In_Leak_$\mathcal{L}$(u)(uid_U(u))
25.   ⊕ attr_In_Flow_$\mathcal{L}$(u′)(uid_U(u)) end
```

2.5.3 Open Routes.

26. A route, r, is open
    - a if all valves, v, of the route are open and
    - b if all pumps, p, of the route are pumping.

```
value
26. is_open: R → Bool
26. is_open(r) ≡
26a.   ∀ mkPu(p):Pu • mkPu(p) ∈ elems r
26a.      ⇒ is_pumping(p) ∧
26b.   ∀ mkVa(v):Va • mkVa(v) ∈ elems r
26b.      ⇒ is_open(v)
```

2.6 Domain Perdurants

2.6.1 Actions.

We shall not formalise any specific actions. Informal examples of actions are: opening and closing a well, start and stop pumping, open and close valves, opening and closing a sink and sense current unit flow.

2.6.2 Events.

We shall not formalise any specific events. Informal examples of events are: empty well, full sink, start pumping signal to pump with no liquid material, pump ignores start/stop pumping signal, valve ignores opening/closing signal, excessive to catastrophic unit leak, and unit fire or explosion.

2.6.3 Behaviours.

We shall not formalise any specific behaviours. Informal examples of behaviours are: start pumping and opening up valves across a pipeline system, and stop pumping and closing down valves across a pipeline system.

3 Basic Domain Description

In this section and in Sect. 5 we shall survey basic principles of describing, respectively, domain intrinsics and other domain facets.

By an **entity** we shall understand a phenomenon that can be observed, i.e., be seen or touched by humans, or that can be conceived as an abstraction of an entity •

Example: Pipeline systems, units and materials are entities (Page 2, Item 1.) ■

The method can thus be said to provide the *domain analysis prompt*: `is_entity` where `is_entity(`θ`)` holds if θ is an entity.

A **domain** is characterised by its observable, i.e., manifest *entities* and their *qualities* •

By a **quality** of an entity we shall understand a property that can be given a *name* and whose *value* can be precisely measured by physical instruments or otherwise identified •

Example: **Unique identifiers** (Page 3, Item 5.), **mereology** (Page 3, Item 8.)and the well capacity (Page 5, Item 20a.), pipe length (Page 5, Item 20b.), current pump height (Page 5, Item 20c.), current valve open/close status (Page 5, Item 20d.) and flow (Page 5, Item 20e.) **attributes** are qualities ■

By a **sort** (or **type** – which we take to be the same) we shall understand the largest set of entities all of which have the same qualities •

By an **endurant entity** (or just, an endurant) we shall understand anything that can be observed or conceived, as a "complete thing", at no matter which given snapshot of time. Thus the method provides a *domain analysis prompt*: `is_endurant` where `is_endurant(`e`)` holds if entity e is an endurant.

By a **perdurant entity** (or just, an perdurant) we shall understand an entity for which only a fragment exists if we look at or touch them at any given snapshot in time, that is, were we to freeze time we would only see or touch a fragment of the perdurant • Thus the method provides a *domain analysis prompt*: `is_perdurant` where `is_perdurant(`e`)` holds if entity e is a perdurant.

By a **discrete endurant** we shall understand something which is separate or distinct in form or concept, consisting of distinct or separate parts • Thus the method provides a *domain analysis prompt*: `is_discrete` where `is_discrete(`e`)` holds if entity e is discrete.

By a **continuous endurant** we shall understand something which is prolonged without interruption, in an unbroken series or pattern • We use the term **material** for continuous endurants • Thus the method provides a *domain analysis prompt*: `is_continuous` where `is_continuous(`e`)` holds if entity e is a continuous entity.

3.1 Endurant Entities

We distinguish between endurant and perdurant entities.

Parts and Materials: The manifest entities, i.e., the endurants, are called parts, respectively materials. We use the term **part** for discrete endurants, that is: `is_part(`p`)`≡ `is_endurant(`p`)`∧`is_discrete(`p`)` • We use the term **material** for continuous endurants •

Discrete endurants are either atomic or are composite.

By an **atomic endurant** we shall understand a discrete endurant which in a given context, is deemed to *not* consist of meaningful, separately observable proper sub-parts • The method can thus be said to provide the *domain analysis prompt*: `is_atomic` where `is_atomic(`p`)` holds if p is an atomic part.

Example: Pipeline units, U, and the monitor, M, are considered atomic ■

By a **composite endurant** we shall understand a discrete endurant which in a given context, is deemed to *indeed* consist of meaningful, separately observable proper sub-parts • The method can thus be said to provide the *domain analysis prompt*: `is_composite` where `is_composite(`p`)` holds if p is a composite part.

Example: The pipeline system, PLS, and the set, Us, of pipeline units are considered composite entities ■

3.1.1 Part Observers.

From atomic parts we cannot observe any sub-parts. But from composite parts we can. For composite parts, p, the *domain description prompt* `observe_part_sorts(`p`)` yields some *formal description text* according to the following *schema*:

```
type  P₁, P₂, ..., Pₙ;[31]
value obs_P₁: P→P₁,
      obs_P₂: P→P₂,
      ...,
      obs_Pₙ: P→Pₙ;[32]
```

where sort names P_1, P_2, ..., P_n are chosen by the domain analyser, must denote disjoint sorts, and may have been defined already, but not recursively A proof obligation may need be discharged to secure disjointness of sorts.

Example: Three formula lines (Page 2, Items 1.) illustrate the basic sorts (PLS′, US, U, M) and observers (obs_US, obs_M) of pipeline systems ■

3.1.2 Sort Models.

A part sort is an abstract type. Some part sorts, P, may have a concrete type model, T. Here we consider only two such models: one model is as sets of parts of sort A: T = A-**set**; the other model has parts being of either of two or more alternative, disjoint sorts: T=P1|P2|...|PN. The *domain analysis prompt*: `has_concrete_type(`p`)` holds if part p has a concrete type. In this case the *domain description prompt* `observe_concrete_type(`p`)` yields some *formal description text* according to the following *schema*,

∗ either

[31]This RSL **type** clause defines P_1, P_2, ..., P_n to be sorts.

[32]This RSL **value** clause defines n function values. All from type P into some type P_i.

> **type**
> $\quad$ P1, P2, ..., PN,
> $\quad$ T = $\mathcal{E}$(P1,P2,...,PN)[33]
> **value**
> $\quad$ **obs_**T: P $\rightarrow$ T[34]

where $\mathcal{E}$(...) is some type expression over part sorts and where P1,P2,...,PN are either (new) part sorts or are auxiliary (abstract or concrete) types[35];

* or:

> **type**
> $\quad$ T = P1 | P2 | ... | PN[36]
> $\quad$ P_1, P_2, ..., P_n
> $\quad$ P1 :: mkP1(P_1),
> $\quad$ P2 :: mkP2(P_2),
> $\quad$ ...,
> $\quad$ PN :: mkPN(P_n) [37]
> **value**
> $\quad$ **obs_**T: P $\rightarrow$ T[38]

Example: **obs_**T: P $\rightarrow$ T is exemplified by obs_Us: PS $\rightarrow$ **U-set** (Page 2, Item 1.), T = P1 | P2 | ... | PN by We | Pu | Va | Fo | Jo | Si (Page 2, Item 3.) and P1 :: mkP1(P_1), P2 :: mkP2(P_2), ..., PN :: mkPN(P_n) by (Page 2, Item 4.) ■

3.1.3 Material Observers.

Some parts p of sort P may contain material. The *domain analysis prompt* `has_material`(p) holds if composite part p contains one or more materials. The *domain description prompt* `observe_material_sorts`(p) yields some *formal description text* according to the following *schema*:

> **type** M_1, M_2, ..., M_m;
> **value obs_**M_1: P $\rightarrow$ M_1, **obs_**M_2: P $\rightarrow$ M_2,
> ..., **obs_**M_m: P $\rightarrow$ M_m;

where values, m_i, of type M_i satisfy `is_material`(m) for all i; and where M_1, M_2, ..., M_m must be disjoint sorts.

Example: We refer to Sect. 2.4 (Page 4, Item 19.) ■

3.2 Endurant Qualities

We have already, above, treated the following properties of endurants: `is_discrete`, `is_continuous`, `is_atomic`, `is_composite` and `has_material`. We may think of those properties as external qualities. In contrast we may consider the following internal qualities: `has_unique_identifier` (parts), `has_mereology` (parts) and `has_attributes` (parts and materials).

[33]The concrete type definition T = $\mathcal{E}$(P1,P2,...,PN) define type T to be the set of elements of the type expressed by type expression $\mathcal{E}$(P1,P2,...,PN).

[34]**obs_T** is a function from any element of P to some element of T.

[35]The *domain analysis prompt*: `sorts_of`(t) yields a subset of {P1,P2,...,PN}.

[36]A|B is the union type of types A and B.

[37]Type definition A :: mkA(B) defines type A to be the set of elements mkA(b) where b is any element of type B

[38]**obs_T** is a function from any element of P to some element of T.

3.2.1 Unique Part Identifiers.

Without loss of generality we can assume that every part has a unique identifier[39]. A **unique part identifier** (or just unique identifier) is a further undefined, abstract quantity. If two parts are claimed to have the same unique identifier then they are identical, that is, their possible mereology and attributes are (also) identical • The *domain description prompt*: `observe_unique_identifier`(p) yields some *formal description text* according to the following *schema*:

> **type** PI;
> **value uid_**P: P $\rightarrow$ PI;

Example: We refer to Page 3, Item 5. ■

3.2.2 Part Mereology.

By **mereology** (Luschei 1962) we shall understand the study, knowledge and practice of parts, their relations to other parts and "the whole" •

Part relations are such as: two or more parts being connected, one part being embedded within another part, and two or more parts sharing attributes.

The *domain analysis prompt*: `has_mereology`(p) holds if the part p is related to some others parts ($p_a, p_b, \ldots, p_c$). The *domain description prompt*: `observe_mereology`(p) can then be invoked and yields some *formal description text* according to the following *schema*:

> **type** MT = $\mathcal{E}$(PI_A,PI_B,...,PI_C);
> **value mereo_**P: P $\rightarrow$ MT;

where $\mathcal{E}$(...) is some type expression over unique identifier types of one or more part sorts. Mereologies are expressed in terms of structures of unique part identifiers. Usually mereologies are constrained. Constraints express that a mereology's unique part identifiers must indeed reference existing parts, but also that these mereology identifiers "define" a proper structuring of parts.

Example: We refer to Items 8.–8g. Pages 3–3 ■

3.2.3 Part and Material Attributes.

Attributes are what really endows parts with qualities. The external properties `is_discrete`,`is_continuous`,`is_atomic`,`is_composite has_material`. are far from enough to distinguish one sort of parts from another. Similarly with unique identifiers and the mereology of parts. We therefore assume, without loss of generality, that every part, whether discrete or continuous, whether, when discrete, atomic or composite, has at least one attribute.

By an **endurant attribute**, we shall understand a property that is associated with an endurant e of sort E, and if removed from endurant e, that endurant would no longer be endurant e (but may be an endurant of some other sort E'); and where that property itself has no physical extent (i.e., volume), as the endurant may have, but may be measurable by physical means • The *domain description prompt* `observe_attributes`(p) yields some *formal description text* according to the following *schema*:

> **type** A_1, A_2, ..., A_n;
> **value attr_**A_1:P$\rightarrow$$A_1$,
> $\quad$ **attr_**A_2:P$\rightarrow$$A_2$,
> $\quad$ ...,
> $\quad$ **attr_**A_n:P$\rightarrow$$A_n$;

Example: We refer to Sect. 2.5 Pages 5–5 ■

[39]That is, `has_unique_identifier`(p) for all parts p.

3.3 Perdurant Entities

We shall not cover the principles, tools and techniques for "discovering", analysing and describing domain actions, events and behaviours to anywhere the detail with which the "corresponding" principles, tools and techniques were covered for endurants. But we shall summarise one essence for the description of perdurants.

There is a notion of **state**. Any composition of parts having dynamic qualities can form a state. Dynamic qualities are qualities that may change. Examples of such qualities are the mereology of a part, and part attributes whose value may change.

There is the notion of **function signature**. A function signature, f: A $(\rightarrow|\overset{\sim}{\rightarrow})$ R, gives a name, say f, to a function, expresses a type, say T_A, of the arguments of the function, expresses whether the function is total $(\rightarrow)$ or partial $(\overset{\sim}{\rightarrow})$, and expresses a type, say T_R, of the result of the function.

There is the notion of **channel**s of synchronisation & communication between behaviours. Channels have names, e.g., ch, ch_i, ch_o. Channel names appear in the signature of behaviour functions: **value** b: A $\rightarrow$ **in** ch_i **out** ch_o R. **in** ch_i indicates that behaviour b may express willingness to communicate an input message over channel ch_i; and **out** ch_o indicates that behaviour b may express an offer to communicate an output message over channel ch_o.

There is a notion of **function pre/post-conditions**. A function pre-condition is a predicate over argument values. A function post-condition is a predicate over argument and result values.

Action signatures include states, Σ, in both arguments, $\mathsf{A}\times\Sigma$, and results,Σ: f: $\mathsf{A}\times\Sigma\rightarrow\Sigma$; f denotes a function in the function space $\mathsf{A}\times\Sigma\rightarrow\Sigma$. Action pre/post-conditions:

```
value
    f(a,$\sigma$) as $\sigma'$;
    pre: $\mathcal{P}_f$(a,$\sigma$);
    post: $\mathcal{Q}_f$(a,$\sigma$,$\sigma'$)
```

have predicates $\mathcal{P}_f$ and $\mathcal{Q}_f$ delimit the value of f within that function space.

Event signatures are typically predicates from pairs of before and after states: e: $\Sigma\times\Sigma\rightarrow\mathbf{Bool}$. Event pre/post-conditions

```
value
    e:$\Sigma \times\Sigma\rightarrow\mathbf{Bool}$;
    e($\sigma$,$\sigma'$) $\equiv$
        $\mathcal{P}_e(\sigma) \wedge \mathcal{Q}_e(\sigma,\sigma')$
```

have predicates $\mathcal{P}_e$ and $\mathcal{Q}_e$ delimit the value of e within the$\Sigma\ \times\Sigma\rightarrow\mathbf{Bool}$ function space; $\mathcal{P}_e$ characterises states leading to event e; $\mathcal{Q}_e$ characterises states, σ', resulting from the event caused by σ.

In principle we can associate a behaviour with every part of a domain. Parts, p, are characterised by their unique identifiers, pi:PI and a state, attrs:ATTRS. We shall, with no loss of generality, assume part behaviours to be never-ending. The unique part identifier, pi:PI, and its part mereology, say $\{\mathsf{pi}_1,\mathsf{pi}_2,\ldots,\mathsf{pi}_n\}$, determine a number of channels $\{\mathsf{chs}[\,\mathsf{pi},\mathsf{pi}_j\,]|\mathsf{j}:\{1,2,\ldots,n\}\}$ able to communicate messages of **type** M. Behaviour signatures:

$$\text{b: pi:PI}\times\text{ATTR}\rightarrow\textbf{in}\ \text{in\_chs}\ \textbf{out}\ \text{out\_chs}\ \textbf{Unit}$$

then have input channel expressions in_chs and output channel expressions out_chs be suitable predicates over $\{\mathsf{chs}[\,\mathsf{pi},\mathsf{pi}_j\,]|\mathsf{j}:\{1,2,\ldots,n\}\}$. **Unit** designate that b denote a never-ending process. We omit dealing with behaviour pre-conditions and invariants.

4 Interlude

We have covered one aspect of the modelling of one set of domain entities, the intrinsic facets of endurants. For the modelling of perdurants we refer to (Bjørner 2010*b*, 2011*a*, 2014*a*). In the next section, Sect. 5, we shall survey the modelling of further domain facets. We shall accompany this survey to a survey of safety issues. To do so in a reasonably coherent way we need establish a few concepts: the *safety* notions of *failure*, *error* and *fault*; the notion of *stake-holder* and the notion of *requirements*.

4.1 Safety-related Concepts

Some characterisations are:

Safety: By *safety*, in the context of a domain being dependable, we mean some measure of continuous delivery of service of either correct service, or incorrect service after benign failure, that is: measure of time to catastrophic failure.

Failure: A domain *failure* occurs when the delivered service deviates from fulfilling the domain function, the latter being what the domain is aimed at (Randell 2003).

Error: An *error* is that part of a domain state which is liable to lead to subsequent failure. An error affecting the service is an indication that a failure occurs or has occurred (Randell 2003).

Fault: The adjudged (i.e., the 'so-judged') or hypothesised cause of an error is a *fault* (Randell 2003).

Hazard: A **hazard** is any source of potential damage, harm or adverse health effects on something or someone under certain conditions at work. Hazards are thus domain faults or are faults of the environment of the domain.

Risk: A **risk** is the chance or probability that a person will be harmed or experience an adverse health effect if exposed to a hazard. It may also apply to situations with property or equipment loss.

4.2 Domain versus System Safety

We must reiterate that we are, in this paper, concerned only with issues of domain safety. Usually safety criticality is examined in the context of (new) systems design. When considering domain safety issues we are concerned with hazards of domain entities without any consideration of whether these hazards enter or do not enter into conceived systems.

4.3 Stake-holder

By a **domain stake-holder** we shall understand a person, or a group of persons, "united" somehow in their common interest in, or dependency on the domain; or an institution, an enterprise, or a group of such, (again) characterised (and, again, loosely) by their common interest in, or dependency on the domain •

Examples: The following are examples of pipeline stake-holders: the owners of the pipeline, the oil or gas companies using the pipeline, the pipeline managers and workers, the owners and neighbours of the lands occupied by the pipeline, the citizens possibly worried about gas- or oil pollution, the state authorities regulating and overseeing pipelining, etcetera ■

5 Domain Facets and Safety Criticality

5.1 Introductory Notions

By a **domain facet** we shall understand one amongst a finite set of generic ways of analysing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain •

We shall in this paper distinguish between the following facets: *intrinsics*, *support technologies*, *human behaviour*, *rules &*[40] *regulations* and *organisation & management*.

In the following we refer to respective subsections of (Bjørner 2010*a*) should the reader wish further elaborations of the facet concept.

5.2 Intrinsics

By **domain intrinsics** (Bjørner 2010*a*, 1.4.1, 11–15)[41] we shall understand those phenomena and concepts of a domain which are basic to any of the other facets (listed earlier and treated, in some detail, below), with such domain intrinsics initially covering at least one specific, hence named, stake-holder view •

Example: The example of Sect. 2 focused on the intrinsics of pipeline systems as well as some derived concepts (routes etc.) ■

Hazards: The following are examples of hazards based sôlely on the intrinsics of the domain: environmental hazards: destruction of one or more pipeline units due to an earth quake, an explosion, a fire or something "similar" occurring in the immediate neighbourhood of these units; design faults: the pipeline net is not acyclic; etcetera ■

Intrinsics hazards are such which violate the well-formedness of the domain. A "domain description" is presented, but it is not a well-formed domain description. One could claim that whichever (event) falls outside the intrinsics domain description, whether it violates well-formedness criteria for domain parts or action, event or behaviour pre/post-conditions, is a hazard. In the context of system safety we shall take the position that explicitly identified hazards must be described, also formally.[42]

5.3 Support Technologies

By domain **support technology** (Bjørner 2010*a*, 1.4.2, 15–17) we shall understand technological ways and means of implementing certain observed phenomena or certain conceived concepts •

The facet of support technology, as a concept, is related to actions of specific parts; that is, a part may give rise to one or more support technologies, and we say that the support technologies 'reside' in those parts.

Examples: wells are, in the intrinsics facet description abstracted as atomic units but in real instances they are complicated (composite) entities of pumps, valves and pipes; pumps are similarly, but perhaps not as complicated complex units; valves likewise; and sinks are, in a sense, the inverse of wells ■

Hazards: a pump may fail to respond to a *stop pump* signal; and a valve may fail to respond to an *open valve* signal ■ I think it is fair to say that most papers on the design of safety critical software are on software for the monitoring & control of support technology.

Describing causes of errors is not simple. With today's formal methods tools and techniques[43] quite a lot can be formalised — but not all!

5.4 Human Behaviour

A proper domain description includes humans as both (usually atomic) parts and the behaviours that we (generally) "attach" to parts.

Examples: The human operators that operate wells, valves, pumps and sinks; check on pipeline units; decide on the flow of material in pipes, etcetera ■

By domain **human behaviour** (Bjørner 2010*a*, 1.4.6, 27–29) we shall understand any of a quality spectrum of humans[44] carrying out assigned work: from (i) *careful, diligent* and *accurate,* via (ii) *sloppy* dispatch, and (iii) *delinquent* work, to (iv) outright *criminal* pursuit •

Typically human behaviour focus on actions and behaviours that are carried out by humans. The intrinsics description of actions and behaviours focus sôlely on intended, careful, diligent and accurate performance.

Hazards: This leaves "all other behaviours" as hazards! Proper hazard analysis, however, usually explicitly identifies failed human behaviours, for example, as identified deviations from described actions etc. Hazard descriptions thus follow from "their corresponding" intrinsics descriptions ■

5.5 Rules & Regulations

Rules and regulations (Bjørner 2010*a*, 1.4.4, 24–26) come in pairs $(\mathcal{R}_u, \mathcal{R}_e)$.

5.5.1 Rules.

By a domain **rule** we shall understand some text which prescribes how people are, or equipment is, "expected" (for "..." see below) to behave when dispatching their duty, respectively when performing their function •

Example: There are rules for operating pumps. One is: A pump, p, on some well-to-sink route $r = r'^\frown\langle p\rangle^\frown r''$, may not be started if there does not exist an open, embedded route r''' such that $\langle p\rangle^\frown r'''$ ends in an open sink ■

Hazards: when stipulating "expected", as above, the rules more or less implicitly express also the safety criticality: that is, when people are, or equipment is, behaving erroneously ■

Example: A domain rule which states, for example, that a pump, p, on some well-to-sink route

[40]We use the ampersand '&' between terms A and B to emphasize that we mean to refer to one subject, the conjoint $A\&B$

[41](Bjørner 2010*a*, 1.4.1, 11–15) refers to publication (Bjørner 2010*a*), Sect. 1.4.1, Pages 11–15.

[42]We refer to the example of Sect. 2. More specifically to the well-formedness of pipeline systems as expressed in wf_PLS (Page 2, Item 2.). We express hazards of the intrinsics of pipeline systems by named predicates over PLS′ and not PLS.

[43]These tools and techniques typically include two or more formal specification languages, for example: **VDM** (Bjørner & Jones 1978, 1982, Fitzgerald & Larsen 1998), **DC** (Zhou & Hansen 2004), **Event-B** (Abrial 2009*a*), **RAISE/RSL** (George et al. 1995, 1992, Bjørner 2006*a,b,c*), **TLA+** (Lamport 2002) and **Alloy** (Jackson 2006); one or more theorem proving tools, for example: **ACL** (Kaufmann et al. 2000*b,a*), **Coq** (Bertot & Castéran 2004), **Isabelle/HOL** (Nipkow et al. 2002), **STeP** (Bjørner et al. 2000), **PVS** (Shankar et al. 1999) and **Z3** (Bjørner et al. 2013); a model-checker, for example: **SMV** (Clarke et al. January 2000) and **SPIN/Promela** (Holzmann 2003); and other such tools and techniques; cf. (Bjørner & Havelund 2014).

[44]— in contrast to technology

$r = r'^\frown\langle p\rangle^\frown r''$, may be started even if there does not exist an open, embedded route r''' such that $\langle p\rangle^\frown r'''$ ends in an open sink is a hazardous rule ■

Modelling Rules: We can model a rule by giving it both a syntax and a semantics. And we can choose to model the semantics of a rule, $\mathbb{R}_u$, as a predicate, $\mathcal{P}$, over pairs of states: $\mathcal{P} : \Sigma\times\Sigma\to\mathbf{Bool}$. That is, the meaning, $\mathcal{M}$, of $\mathbb{R}_u$ is $\mathcal{P}$. An action or an event has changed a state σ into a state σ'. If $\mathcal{P}(\sigma,\sigma')$ is **true** it shall mean that the rule as been obeyed. If it is **false** it means that the rule has been violated.

5.5.2 Regulations.

By a domain **regulation** we shall understand some text which "prescribe" ("…", see below) the remedial actions that are to be taken when it is decided that a rule has not been followed according to its intention •

Example: There are regulations for operating pumps and valves: Once it has been discovered that a rule is hazardous there should be a regulation which (i) starts an administrative procedure which ensures that the rule is replaced; and (ii) starts a series of actions which somehow brings the state of the pipeline into one which poses no danger and then applies a non-hazard rule ■

Hazards: when stipulating "prescribe", regulations express requirements to emerging hardware and software ■

Modelling Regulations: We can model a regulation by giving it both a syntax and a semantics. And we can choose to model the semantics of a regulation, $\mathbb{R}_e$, as a state-transformer, $\mathcal{S}$, over pairs of states: $\mathcal{S} : \Sigma\times\Sigma\to\Sigma$. That is, the meaning, $\mathcal{M}$, of $\mathbb{R}_e$ is $\mathcal{S}$. A state-transformation $\mathcal{S}(\sigma,\sigma')$ for rule $\mathbb{R}_u$ results in a state σ'' where: if $\mathcal{P}(\sigma,\sigma')$ is **true** then $\sigma' = \sigma''$, else σ'' is a corrected state such that $\mathcal{P}(\sigma,\sigma'')$ is **true**.

5.5.3 Discussion.

Where do rules & regulations reside?" That is, *"Who checks that rules are obeyed?"* and *"Who ensures that regulations are applied when rules fail?"* Are some of these checks and follow-ups relegated to humans (i.e., parts) or to machines (i.e., "other" parts)? that is, to the behaviour of part processes? The next section will basically answer those questions.

5.6 Organisation & Management

To (Bjørner 2010*a*, 1.4.3, 17–21) properly appreciate this section we need remind the reader of concepts introduced earlier in this paper. With parts we associate mereologies, attributes and behaviours. Support technology is related to actions and these again focused on parts. Humans are often modelled first as parts, then as their associated behaviour. It is out of this seeming jigsaw puzzle of parts, mereologies, attributes, humans, rules and regulations that we shall now form and model the concepts of organisation and management.

5.6.1 Organisation.

By domain **organisation** we shall understand one or more partitionings of resources where resources are usually representable as parts and materials and where usually a resource belongs to exactly one partition; such that n such partitionings typically reflects strategic[45] (say partition π_s), tactical[46] (say partition π_t), respectively operational [47] (say partition π_o) concerns (say for $n = 3$), and where "descending" partitions, say π_s, π_t, π_o, represents *coarse, medium* and *fine* partitions, respectively •

Examples: This example only illustrates production aspects. At the strategic level one may partition a pipeline system into just one component: the entire collection of all pipeline units, π. At the tactical level one may further partition the pipeline system into the partition of all wells, π_{ws}, the partition of all sinks, π_{ss}, and a partition of all pipeline routes, $\pi_{\ell s}$, that $\pi_{\ell s}$, is the set of all routes of π excluding wells and sinks. At the organisational level may further partition the pipeline system into the partitions of individual wells, π_{w_i} ($\pi_{w_i} \in \pi_{ws}$), the partitions of individual sinks, π_{s_j} ($\pi_{s_i} \in \pi_{ws}$) and the partitions of individual pipeline routes, π_{r_k} ($\pi_{\ell_i} \in \pi_{\ell s}$) ■

A domain organisation serves to structure management and non-management staff levels and the allocation of strategic, tactical and operational concerns across all staff levels; and hence the "lines of command": who does what, and who reports to whom, administratively and functionally.

Organisations are conceptual parts, that is, partitions are concepts, they are conceptual parts in addition, i.e., adjoint to physical parts. They serve as "place-holders" for management.

Modelling Organisations: We can normally model an organisation as an attribute of some, usually composite, part. Typically such a model would be in terms of the one or more partitionings of unique identifiers, π:Π, of domain parts, p:P. For example:

```
type
    ORG = Str × Tac × Ope × ...
    Str, Tac, Ope = (Π-set)-set
value
    attr_ORG: P → ORG
axiom
    $\mathcal{P}$: ORG → ... → Bool
```

where we leave the details of the partitionings Str, Tac, Org, … and the axiom governing the individual partitionings and their relations for further analysis.

5.6.2 Management.

By domain **management** we shall understand such people who (such decisions which) (i) determine, formulate and thus set standards (cf. rules and regulations, above) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management, and to floor staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who "backstops" complaints from lower management levels and from floor staff •

Example: At the strategic level there is the overall management of the pipeline system. At the tactical level there may be the management of all wells;

[45] Strategic management, one can claim, deals with the management of the most generic and general, year-to-year company resources: invested capital, overall market, production and service goals, etc.

[46] Tactical management, one can claim, deals with the management of the quarter/month-to-quarter/month resources "closest" to the implementation if strategic goals.

[47] Operational management, one can finally claim, deals with the management of day-to-day resources "closest" to the actual market, production and services.

all sinks; specific (disjoint) routes. At the operational there may then be the management of individual wells, individual sinks, and individual groups of valves and pumps ■

Modelling Management: Some parts are associated with strategic management. They will have their unique identifiers, $\pi : \Pi$, belong to some partition in an str:Str. Other parts are associated with tactical management. They will have their unique identifiers, $\pi : \Pi$, belong to some partition in a corresponding tac:Tac. Yet other parts are associated with operational management. They will have their unique identifiers, $\pi : \Pi$, belong to some partition in the corresponding ope:Ope. The "management" parts have their attributes form corresponding states (σ:Σ).

```
type
    $\Sigma_{STR}$, $\Sigma_{TAC}$, $\Sigma_{OPE}$,
```

An idealised rendition of management actions is:

```
value
    $\text{action}_{Strategic}$: $\Sigma_{STR} \to \Sigma_{TAC} \to \Sigma_{OPE} \to \Sigma_{STR}$
    $\text{action}_{Tactical}$: $\Sigma_{STR} \to \Sigma_{TAC} \to \Sigma_{OPE} \to \Sigma_{TAC}$
    $\text{action}_{Operational}$: $\Sigma_{STR} \to \Sigma_{TAC} \to \Sigma_{OPE} \to \Sigma_{OPE}$
```

$\text{action}_{Strategic}$ expresses that strategic management considers the "global" state ($\Sigma_{STR} \times \Sigma_{TAC} \times \Sigma_{OPE}$) but potentially changes only the "strategy" state.

$\text{action}_{Tactical}$ expresses that tactical management considers the "global" state ($\Sigma_{STR} \times \Sigma_{TAC} \times \Sigma_{OPE}$) but potentially changes only the "tactical" state.

$\text{action}_{Operational}$ expresses that tactical management considers the "global" state ($\Sigma_{STR} \times \Sigma_{TAC} \times \Sigma_{OPE}$) but potentially changes only the "operational" state.

We can normally model management as part of the behavioural model of some, usually composite part. Typically such a model would be in terms communication procedures between managers, p:P, and their immediate subordinates, $\{p_1:P_1, p_2:P_2, \ldots, p_n:P_N\}$: For example:

```
channel mgt:$\{\{\pi,\pi_j\}|\pi_j:\text{PI}_j \bullet \pi_j \in ...\}$:M
value
    p: $\pi:\Pi \times$ pt:P $\to$
            in,out $\{\{\pi,\pi_j\}|\pi_j:\text{PI}_j \bullet \pi_j \in ...\}$  Unit
    p($\pi$,pt) $\equiv$ ...
              [ management orders staff ]
       $\lceil\rceil$ let ($\pi_j$,m) = $\text{query}_{\text{boss}}$(p) in
            m ! mgt[$\{\pi,\pi_j\}$]!m ;
            p($\pi$,$\text{action}_{down_s}$(pt,m)) end
              [ management "listens" to staff ]
       $\lceil\rceil$ let ($\pi_j$,m) = $\lceil\rceil$$\{$ mgt[$\{\pi,\pi_j\}$]? | ... $\}$ in
            p($\pi$,$\text{action}_{down_r}$(pt,m)) end
              [ management reports to boss ]
       $\lceil\rceil$ let ($\pi_{\text{boss}}$,m) = $\text{query}_{\text{staff}}$(pt) in
            m ! mgt[$\{\pi,\pi_{\text{boss}}\}$]!m ;
            p($\pi$,$\text{action}_{up_s}$(pt,m)) end
              [ management "listens" to boss ]
       $\lceil\rceil$ let ($\pi_{\text{boss}}$,m) =
                 $\lceil\rceil$$\{$ mgt[$\{\pi,\pi_{\text{boss}}\}$]? | ... $\}$ in
            p($\pi$,$\text{action}_{up_s}$(pt,m)) end ...
```

The boss communications express that process p serves a boss. All other communications express that process p interacts with staff (i.e., "subordinates and "others").

Hazards: Hazards of organisations & management come about also as the result of "mis-management": Strategic management updates tactical and operational management states. Tactical management updates strategic and operational management states. Operational management updates strategic and tactical management states. That is: these states are not clearly delineated, Etcetera!

● ● ●

This section on organisation & management is rather terse; in fact it covers a whole, we should think, novel and interesting theory of business organisation & management.

5.7 Discussion

There may be other facets but our point has been made: that an analysis of hazards (including faults) can, we think, be beneficially structured by being related to reasonably distinct facets.

A mathematical explanation of the concept of facet is needed. One that helps partition the domain phenomena and concepts into disjoint descriptions. We are thinking about it and encourage the reader to do likewise!

6 Conclusion

The present author's research has since the early 1970s focused on programming methodology: how to develop software such that it was correct with respect to some specification — call it requirements. The emphasis was on abstract software specifications and their refinement or transformation into code. Programming language semantics and the stage- and step-wise development of compilers, in many, up to nine stages and steps, became a highlight of the 1980s. The step from programming language semantics to domain descriptions followed: Domain descriptions, in a sense, specified the language inherent in the described domain — that is: "spoken" by its actors, etc. Since the early 1990s I therefore additionally focused on domain descriptions. Now an additional goal of software development might be achieved: securing that the software meet customers' expectations. With the observation (Bjørner 2008) that requirements prescriptions can be systematically — but, of course, not automatically — "derived" from domain descriptions a bridge was established: from domains via requirements to software.

6.1 Comparison to Other Work

(Bjørner 2014*b*) contains a large section, Sect. 4.1 (4+ pages), which compares our domain analysis and description approach to the domain analysis approaches of *Ontology and Knowledge Engineering*, *Database Analysis* (Bachmann Diagrams, Relational Data Models, Entity Set Relations, etc., *Prieto-Dĩaz's* work, *Domain Specific Languages*, *Feature-oriented Domain Analysis*, *Software Product Line Engineering*, Michael Jackson's *Problem Frames*, *Domain Specific Software Architectures*, *Domain Driven Design*, *Unified Modelling Language*, etcetera. We refer to (Bjørner 2014*b*) for its lengthy discussion and almost 30 citations. (Bjørner 2014*b*, Sect. 4.1) shows that our approach is significantly different from the *above-enumerated approaches.*

6.2 What Have We Achieved ?

When Dr Clive Victor Boughton, on November 4, 2013, approached me on the subject of *"Software*

Safety: New Challenges and Solutions", I therefore, naturally questioned: can one stratify the issues of safety criticality into three phases: searching for sources of faults and hazards in **domains,** elaborating on these while "discovering" further sources during **requirements** engineering, and, finally, during early stages of **software design.** I believe we have answered that question partially with there being good hopes for further stratification.

Yes, I would indeed claim that we have contributed to the "greater" issues of safety critical systems by suggesting a disciplined framework for faults "discovery"and hazards: investigate separately the domains, the requirements and the design.

6.3 Further Work

But, clearly, that work has only begun.

7 Acknowledgements

I thank Dr Clive Victor Boughton of aSSCa, ANU, &c. for having the courage to convince his colleagues to invite me, for having inspired me to observe that faults and hazards can be "discovered" purely in the context of domain descriptions, for his support in answering my many questions, and for otherwise arranging my visit. I also, with thanks, acknowledge comments and remarks by the ASSC program chair, Dr Anthony Cant and especially by hos colleague Dr Brendan Mahony. Their joint paper (Cant & Mahony 2012), alas, came only to my attention in the last days before the present paper had to be submitted.

8 Bibliography

8.1 Notes

This conference contribution is part of a series of papers on the topic of domains. (Bjørner 2007, 2008, 2010*a*,*b*, 2011*a*,*b*, 2013*a*,*b*, 2014*b*, 2009, 2010*c*, Bjørner & Eir 2010). In (Bjørner 2008) we show how to "derive" requirements prescriptions from domain descriptions; (Bjørner 2010*a*) shows techniques for describing domain facets: intrinsics, support technologies, rules & regulations, management & organisation as well as human behaviour; (Bjørner 2011*b*) illuminates such concepts as simulation, demos, monitoring and control in the new light afforded by the domain viewpoint; (Bjørner 2013*b*) speculates on various issues of "computation for humanity" (!); (Bjørner 2013*a*) relates our modelling of mereology to the classical axiom systems for mereology; and (Bjørner 2014*c*) provides a systematic introduction to principles, techniques and tools for the analysis and description of domain endurants.

8.2 References

Abrial, J.-R. (1996 and 2009*b*), The B Book: Assigning Programs to Meanings *and* Modeling in Event-B: System and Software Engineering, Cambridge University Press, Cambridge, England.

Abrial, J.-R. (2009*a*), *Modeling in Event-B: System and Software Engineering*, Cambridge University Press, Cambridge, England.

Benjamin, J. & Fensel, D. (1998), *The Ontological Engineering Initiative (KA)2.* Internet publication + Formal Ontology in Information Systems, University of Amsterdam, SWI, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands and University of Karlsruhe, AIFB, 76128 Karlsruhe, Germany, 1998.http://www.aifb.uni-karlsruhe.de/WBS/broker/KA2.htm.

Bertot, Y. & Castéran, P. (2004), *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*, EATCS Series: Texts in Theoretical Computer Science, Springer.

Bjørner, D. (2006*a*), *Software Engineering, Vol. 1: Abstraction and Modelling*, Texts in Theoretical Computer Science, the EATCS Series, Springer.

Bjørner, D. (2006*b*), *Software Engineering, Vol. 2: Specification of Systems and Languages*, Texts in Theoretical Computer Science, the EATCS Series, Springer. Chapters 12–14 are primarily authored by Christian Krog Madsen.

Bjørner, D. (2006*c*), *Software Engineering, Vol. 3: Domains, Requirements and Software Design*, Texts in Theoretical Computer Science, the EATCS Series, Springer.

Bjørner, D. (2007), Domain Theory: Practice and Theories, Discussion of Possible Research Topics, *in* 'ICTAC'2007', Vol. 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, Springer, Heidelberg, pp. 1–17.

Bjørner, D. (2008), From Domains to Requirements, *in* 'Montanari Festschrift', Vol. 5065 of *Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer)*, Springer, Heidelberg, pp. 1–30.

Bjørner, D. (2009), *Domain Engineering: Technology Management, Research and Engineering*, A JAIST Press Research Monograph # 4, 536 pages.

Bjørner, D. (2010*a*), Domain Engineering, *in* P. Boca & J. Bowen, eds, 'Formal Methods: State of the Art and New Directions', Eds. Paul Boca and Jonathan Bowen, Springer, London, UK, pp. 1–42.

Bjørner, D. (2010*b*), 'Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*', *Kibernetika i sistemny analiz* (4), 100–116.

Bjørner, D. (2010*c*), The Rôle of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed!, *in* 'Perspectives of Systems Informatics', Vol. 5947 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 2–34.

Bjørner, D. (2011*a*), 'Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*', *Kibernetika i sistemny analiz* (2), 100–120.

Bjørner, D. (2011*b*), Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions, *in* 'Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.', Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), Springer, Heidelberg, Germany, pp. 167–183.

Bjørner, D. (2013*a*), *A Rôle for Mereology in Domain Science and Engineering*, Synthese Library (eds. Claudio Calosi and Pierluigi Graziani), Springer, Amsterdam, The Netherlands.

Bjørner, D. (2013*b*), *Domain Science and Engineering as a Foundation for Computation for Humanity*, Computational Analysis, Synthesis, and Design of Dynamic Systems, CRC [Francis & Taylor], chapter 7, pp. 159–177. (eds.: Justyna Zander and Pieter J. Mosterman).

Bjørner, D. (2014*a*), Domain Analysis & Description: Perdurants [Writing to begin Summer/Fall 2014], Research Report, Fredsvej 11, DK-2840 Holte, Denmark.

Bjørner, D. (2014*b*), Domain Analysis: Endurants – An Analysis & Description Process Model, *in* J. Meseguer & K. Ogata, eds, 'Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi', pages 1–34, Springer.

Bjørner, D. (2014*c*), Domain Analysis, Fredsvej 11, DK-2840 Holte, Denmark. (**Note:** This is currently the "definitive" paper on domain description methodology: `www.imm.dtu.dk/~dibj/2014/domain--analysis.pdf`.)

Bjørner, D. & Eir, A. (2010), Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann, *in* 'Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness', Vol. 5930 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 22–59.

Bjørner, D. & Havelund, K. (2014), 40 Years of Formal Methods — 8 Obstacle and 3 Possibilities, *in* 'FM 2014, Singapore, May 14-16, 2014', Springer. Distinguished Lecture.

Bjørner, D. & Jones, C. B., eds (1978), *The Vienna Development Method: The Meta-Language*, Vol. 61 of *LNCS*, Springer.

Bjørner, D. & Jones, C. B., eds (1982), *Formal Specification and Software Development*, Prentice-Hall.

Bjørner, N., Browne, A., Colon, M., Finkbeiner, B., Manna, Z., Sipma, H. & Uribe, T. (2000), 'Verifying Temporal Properties of Reactive Systems: A STeP Tutorial', *Formal Methods in System Design* **16**, 227–270.

Bjørner, N., McMillan, K. & Rybalchenko, A. (2013), Higher-order Program Verification as Satisfiability Modulo Theories with Algebraic Data-types, *in* 'Higher-Order Program Analysis'. http://hopa.cs.rhul.ac.uk/files/proceedings.html.

Cant, A. & Mahony, B. (2012), Safety protocols: a new safety engineering paradigm, *in* Australian System Safety Conference (ASSC 2012).

Clarke, E. M., Grumberg, O. & Peled, D. A. (January 2000), *Model Checking*, The MIT Press, Five Cambridge Center, Cambridge, MA 02142-1493, USA. ISBN 0-262-03270-8.

Fitzgerald, J. & Larsen, P. G. (1998), *Modelling Systems – Practical Tools and Techniques in Software Development*, Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK. ISBN 0-521-62348-0.

George, C. W., Haff, P., Havelund, K., Haxthausen, A. E., Milne, R., Nielsen, C. B., Prehn, S. & Wagner, K. R. (1992), *The RAISE Specification Language*, The BCS Practitioner Series, Prentice-Hall, Hemel Hampstead, England.

George, C. W., Haxthausen, A. E., Hughes, S., Milne, R., Prehn, S. & Pedersen, J. S. (1995), *The RAISE Development Method*, The BCS Practitioner Series, Prentice-Hall, Hemel Hampstead, England.

Fox, C. (2000), *The Ontology of Language: Properties, Individuals and Discourse.* CSLI Publications, Center for the Study of Language and Information, Stanford University, California, ISA, 2000.

Holzmann, G. J. (2003), *The SPIN Model Checker, Primer and Reference Manual*, Addison-Wesley, Reading, Massachusetts.

IEEE Computer Society (1990), IEEE–STD 610.12-1990: Standard Glossary of Software Engineering Terminology, Technical report, IEEE, IEEE Headquarters Office, 1730 Massachusetts Avenue, N.W., Washington, DC 20036-1992, USA. Phone: +1-202-371-0101, FAX: +1-202-728-9614.

Jackson, D. (2006), *Software Abstractions: Logic, Language, and Analysis*, The MIT Press, Cambridge, Mass., USA. ISBN 0-262-10114-9.

Kaufmann, M., Manolios, P. & Moore, J. S. (2000*a*), *Computer-Aided Reasoning: ACL2 Case Studies*, Kluwer Academic Publishers.

Kaufmann, M., Manolios, P. & Moore, J. S. (2000*b*), *Computer-Aided Reasoning: An Approach*, Kluwer Academic Publishers.

Lamport, L. (2002), *Specifying Systems*, Addison–Wesley, Boston, Mass., USA.

Luschei, E. (1962), *The Logical Systems of Leśniewksi*, North Holland, Amsterdam, The Netherlands.

Mellor, D.H. & Oliver, A., editors (1997), *Properties*. Oxford Readings in Philosophy. Oxford Univ Press, May 1997. ISBN: 0198751761, 320 pages.

Nipkow, T., Paulson, L. C. & Wenzel, M. (2002), *Isabelle/HOL, A Proof Assistant for Higher-Order Logic*, Vol. 2283 of *Lecture Notes in Computer Science*, Springer-Verlag.

Randell, B. (2003), On Failures and Faults, *in* 'FME 2003: Formal Methods', Vol. 2805 of *Lecture Notes in Computer Science*, Formal Methods Europe, Springer, pp. 18–39. Invited paper.

Shankar, N., Owre, S., Rushby, J. M. & Stringer-Calvert, D. W. J. (1999), *PVS Prover Guide*, Computer Science Laboratory, SRI International, Menlo Park, CA.

Woodcock, J. C. P. & Davies, J. (1996), *Using Z: Specification, Proof and Refinement*, Prentice Hall International Series in Computer Science. **URL:** *http://www.comlab.ox.ac.uk/usingz.html*

Zhou, C. C. & Hansen, M. R. (2004), *Duration Calculus: A Formal Approach to Real–time Systems*, Monographs in Theoretical Computer Science. An EATCS Series, Springer–Verlag.

Research Papers

Accident Modelling Using Social Network Analysis for Complex Socio-Technical Systems

Karen Klockner & Yvonne Toft
School of Human, Health & Social Sciences
Central Queensland University
160 Ann Street, Brisbane, Queensland

k.klockner@cqu.edu.au; y.toft@cqu.edu.au

Abstract

Prevention of safety occurrences has long been tied to accident modelling, in an attempt to understand how the event happened. Accident modelling has traditionally being based on understanding accidents as a linear sequence of events, however today modern software programs, particularly those being used for Social Network Analysis (SNA), offer the possibility to understand accidents as a complex network of contributing factors, perhaps for the first time. **Aim**: The aim of this research was to understand reoccurring patterns of contributing factors to major safety occurrences through the use of SNA. **Method:** Major railway accident reports were analysed and data collected on the contributing factors for high risk railway accidents for a five year period. The contributing factors were then modelled using SNA software programs. **Results:** The use of SNA has enabled the investigation and understanding of how safety occurrence contributing factors are interlinked (networked) and the contribution that each of the factors has on accident phenomenology in complex socio-technical systems. **Discussion**: The outcome of this project has been the development of a new method of accident modelling that moves beyond traditional linear models and better reflects modern day safety science thinking about complex systems. This type of modelling can be used as a proactive risk management tool by providing the knowledge of which contributing factors need to be controlled or mitigated for accident prevention.

Keywords: Safe-NET, Accident Modelling, Social Network Analysis, Contributing Factors, Complex Socio-Technical Systems

1 Introduction

Accident modelling has long been used as a tool for understanding how a safety incident occurred and has been traditionally limited by the view that accidents are a linear sequence of events which ultimately result in an accident or other safety occurrence.

Early accident modelling provided a simplistic picture or visual representation of how accidents were thought to occur. They demonstrated the 'straight line path' that accidents were thought to take in their development, with each part representing the contributing factors that would have contributed to or occurred during the event. The first of these models is show in Figure 1 being Heinrich's (1931) Domino Theory.

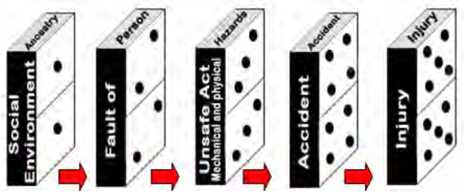


Figure 1 – Heinrich Domino Theory

The early industrial age in which this model was developed is a far cry from the complex socio-technical systems in which modern day accident investigation is conducted. Whilst accident models have certainly evolved since the first model appeared there is still the desire to be able to visually represent what is understood to be happening in today's complex systems through more characteristic models.

Whilst work in the area of System Dynamics (Sterman, 2000) and the resulting casual loop diagrams developed by this method have focussed on understanding a system and its constituent components and their interactions, its value in accident understanding has not been well received, perhaps due to its inherent complexity and whilst coming closer to complex system understanding these models have not fulfilled the current need in the accident modelling of complex systems.

Generally then, the safety professional has been confined to the well used and accepted accident models which have not adequately moved past representations of safety occurrences as a chain of events traditionally tied to linear causality relationships.

This present research represents an attempt to improve accident modelling. The proposed method and resulting models presents a representation of the interconnectedness of safety occurrence 'contributing factors' which exist within complex socio-technical systems. The ability to create new accident models of interconnectedness is now achievable due to current network analysis methods such as Social Network Analysis.

2 Complex Socio Technical Systems

Socio-technical systems refer to systems in which humans and technology interact. They are today complex because of their interconnected nature and the large amount of the constituent parts which are reliant on each other.

Whilst modern day organisations have moved into these complex systems due to changes in technology and the way of work, accident prevention in these systems has been difficult to achieve due to the inability to fully keep up with technology advances, the inability to understand human factors in wider organisational design issues and for the representation of these systems in accident phenomenology through accident modelling.

Whilst accident investigation and the identification of causal/contributing factors is said to have moved to a 'systems approach' (based on systems theory whereby one has to understand the components which work separately as well as understanding how the components work within the larger system in which they operate (Senge, 2006)) the linear accident trajectory model is not consistent with the modern concepts of non-linear systems thinking.

In thinking about complex systems, safety science researchers are beginning to recognise that: (a) events are not clearly distinguished nor are they independent, (b) there is no clear flowing of influence from one event to the next and (c) time is not necessarily linear and unambiguous but can be simultaneously without any time lag (Buzsáki, 2006).

Complex systems in this context then are not implied to mean complicated but the implication is towards a non-linear relationship between the system's components. These complex systems are also termed "open", and whilst they appear stable they are in constant states of change which are seen as a defining feature when compared to 'closed' systems which remain in balance, are simple and hard to perturb (Buzsáki, 2006).

What has been recognised is that safety science thinking is moving towards complex systems thinking. Goh, Brown and Spickett (2010, p. 302) introduce the concept that "current causal analysis tools are not designed to analyse dynamic complexity of major incidents … which arises from the interactions between actors and the temporal and spatial gaps between actions and consequences". These researchers identified that most tools are designed with only linear cause and effect relationships, and there is a need to use and develop tools which are designed to model dynamic complexity such as the causal loop diagrams which are aligned to the recursive nature of most systems. The problem then arises as to how to move forward in describing these recursive relationships (Goh, Brown and Spickett, 2010; Senge, 2006).

The current impetus then is for new methods and models which can cope with system complexity and promote the development of accident models which reflect the complexity of the safety systems relationships which are now understood to exist. Roelen, Lin and Hale (2010, p. 5) have stated that any new accident model "requires the combination of detailed knowledge of all aspects of the system, processing huge amounts of data, a substantial mathematical background and the ability to capture this all in a user friendly software tool to be used by the safety analysts."

At the very least new accident models are required which give a clearer picture of this complexity and can examine and explain the inter-connectiveness of aspects within the system and the relationship of constituent parts as a starting point. A way of modelling accidents is required where cause and effect relationships are free to vary, joining and linking in perhaps previously unconsidered ways. The present research project illustrates a new application of Social Network Analysis to model the complexity of non-linear accidental model relationships.

3 Visualization and Network Analysis

A network is, in its simplest form, a collection of points (nodes) joined together in pairs of lines (edges) (Newman, 2010). Newman identifies that there are various focuses to any study of networks but three main reasons prevail, being (1) a review of the nature of the individual components, (2) to the study of the nature of the connections or interactions; and perhaps most importantly (3) to see and understand the pattern of connections between the components.

For researchers wanting to use network analysis as a research method, modern software programs offer the ability to transform data to enable it to be presented as both a visual network and in the form of key statistics; predominately 'measures of centrality'.

Social networks represent the connections between people, with the nodes representing people, or groups of people and the edges represent some form of interaction (usually social) between the people or actors in a system. Social network analysis (SNA) lends itself to the study of many different types of interactions with connections able to represent any type of connections from friendships to exchange of money, professional relationship or connections between actors/things. SNA has been adapted, however, to analyses that involve nodes which are not exclusively people, but instead other units of analysis in social systems. SNA methodology has been used in this research to model the connections between safety occurrence contributing factors resulting in a new network accident model now known as Safety and Failure Event Network (Safe-NET).

4 Social Network Analysis

The history of SNA dates back to 1933 with the work of Jacob Moreno who is recognised as doing the first work in the field of sociometry, or SNA as it is now known (Newman, 2010).

SNA allows the exploration of visual patterns found within connections of linked entities or things. This network perspective looks at a collection of ties, traditionally amongst people in communication settings (e.g., Google/ Facebook) and creates measurements that describe the location of each person or entity within the structure of all the relationships in the network. The position or location of an entity or group of entities in relation to all the others is the main concern of SNA (Hansen, Shneiderman & Smith, 2001).

Of primary interest for this research was the use of SNA to enhance the understanding of how railway safety occurrence contributing factors were networked and which factors could be identified as the main contributors to various types of railway safety occurrences. The use of SNA was found to reveal these relationships through a resulting visual (accident) model of how the contributing factors were linked, and also allowed the identification of which factors were contributing the most to certain types of incidents as statistical measures of centrality.

The SNA method also enabled what were once stand alone contributing factors (due to inherent ties to sequential linear accident modelling which resulted in separation of the factors) to now be examined using a systemic approach where all of the contributing factors could be examined together without the artificial boundaries of considering just binary relationships between contributing factors.

5 Modelling Complex Systems

The use of information visualisation using SNA for this research made sense given that one of the aims was to examine the interactions of the contributing factors as a network of connections in a larger complex system with a view to having an accident 'model' as a resulting outcome.

The supposition used to conceptualise and operationalise the transition of this research into the SNA realm was that the contributing factors to safety occurrences could also be thought of as "people or actors" and that these actors, when identified as having been part of a safety occurrence as a contributing factor, would have met each other as if they had attended a party but instead they attended a safety occurrence.

The contributing factors could therefore be thought of as individuals who 'knew' and 'met' each other every time they showed up at a safety occurrence (event). If they were present at the event they met all the other factors that also showed up at the event. The data, on which factors showed up at each event and knew/met each of the other factors at that same event, could therefore be investigated using SNA. This would enable an examination of the networking of the factors for each type event under study. Information visualization using SNA as a method offered both a resulting accident model and the understanding of the actual contribution to the safety occurrence to be identified through the examination of centrality measures in SNA.

6 Accident Modelling Using SNA

Detailed here is a brief description of the Safe-NET methodology used to enable this accident modelling technique, using SNA, to be understood as a step by step process. Whilst this method examined railway incident data, the method itself appears to be transferrable to any industry where data on safety occurrences is collected and the contributing factors identified.

6.1 Data Collection

Contributing factors data was firstly collected by reading and applying the Contributing Factors Framework (CFF) (Contributing Factors Framework, 2012) to Major Railway Safety Incident Reports submitted by rail transport operations to the Rail Safety Regulator in Queensland for the years 2006 and 2010. The CFF is a framework developed to assist the rail industry in Australia identify the contributing factors in rail safety investigations and enables the identification of contributing factors across three main headings, being Individual/Team Factors, Technical Factors and Local Conditions & Organisational Factors. A total of 429 incidents were analysed using the CFF and a total of 2090 contributing factors were identified.

This research was initially specifically interested in four types of high risk railway incidents being Collisions, Derailments, Safe Working Breaches and Signals Passed at Danger however the contributing factors for various sub types under the four main incident heading were also able to be analysed using SNA methods.

6.2 Data Conversion to SNA Format

To allow the contributing factors data to be entered into a SNA program it was required to be reformatted. As SNA is based on the connection between several people (or other entities) it was necessary to link contributing factors to each other. The linking or connection of factors was therefore operationalised by viewing each incident as a 'event' with those contributing factors identified as having attended that event as having met each other, being similar to people who meet each other at any social gathering.

Therefore if a contributing factor was identified as being present at a safety occurrence (event) it met all the other contributing factors at that same event. This allowed relationships between contributing factors to be established for each safety occurrence reviewed. Meetings between factors therefore become the basis of the resultant visual model in SNA, who met who at each event and how many times they met over multiple events.

6.3 Data into SNA Programs

Once the data had been converted to a SNA format it was able to be entered into several SNA programs. There is some variation between programs in as much as some provide better graphics than others, whilst others provide more complex statistical output measures. Much like general statistical packages, however, the methodology of SNA is largely shared among different programs. As such, most programs will produce very similar results given the same data.

7 SNA Outputs

Result outcomes in SNA are examined in terms of patterns of connections around an individual (or other entity), in this case a contributing factor, which results in a visualisation displayed as a network accident model as show in Figure 2. Figure 2 shows a SNA accident model output with all contributing factors shown for Railway Yard Derailments. Human error contributing factors are shown in yellow, Local Conditions & Organisational factors are shown in blue and technical failures are shown in green. Figures 3 to 5 show various manipulated versions of the same model all produced in TouchGraph Navigator 2 (TouchGraph Navigator 2, 2013).

.

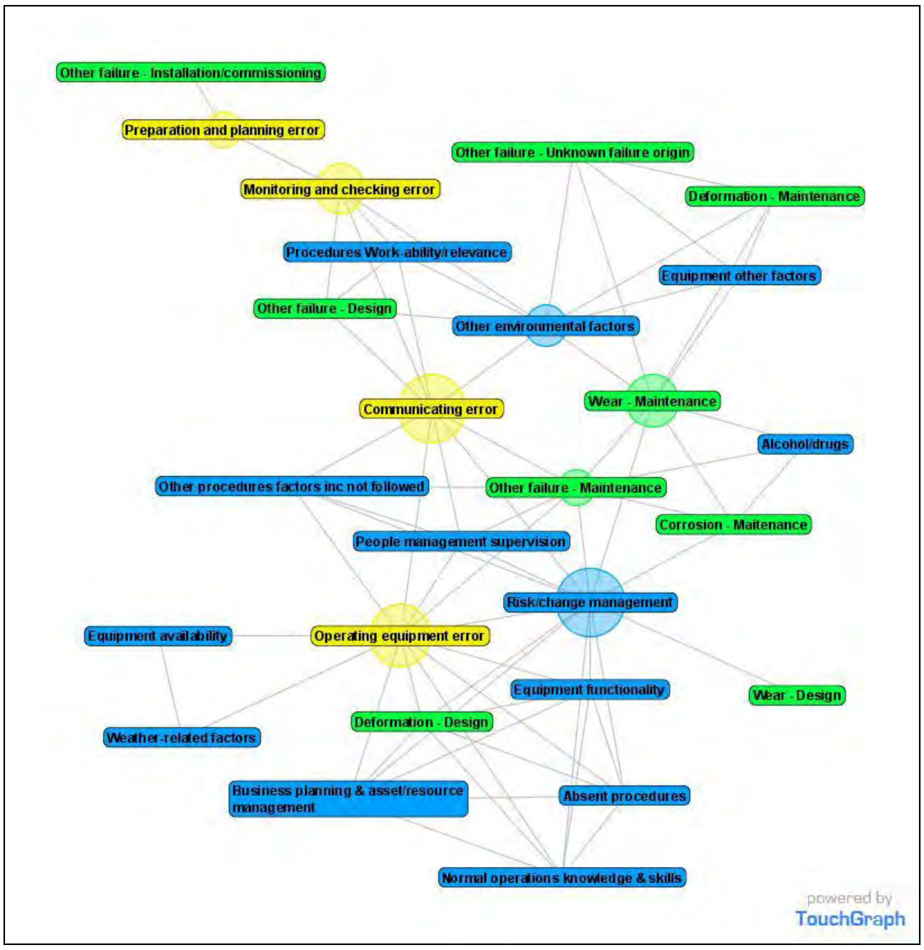

Other failure - Installation/commissioning
Preparation and planning error
Other failure - Unknown failure origin
Monitoring and checking error
Deformation - Maintenance
Procedures Work-ability/relevance
Equipment other factors
Other failure - Design
Other environmental factors
Communicating error
Wear - Maintenance
Alcohol/drugs
Other procedures factors inc not followed
Other failure - Maintenance
Corrosion - Maitenance
People management supervision
Risk/change management
Equipment availability
Operating equipment error
Equipment functionality
Wear - Design
Deformation - Design
Weather-related factors
Business planning & asset/resource management
Absent procedures
Normal operations knowledge & skills
powered by TouchGraph

Figure 2: Railway Yard Derailments – All Contributing Factors Shown

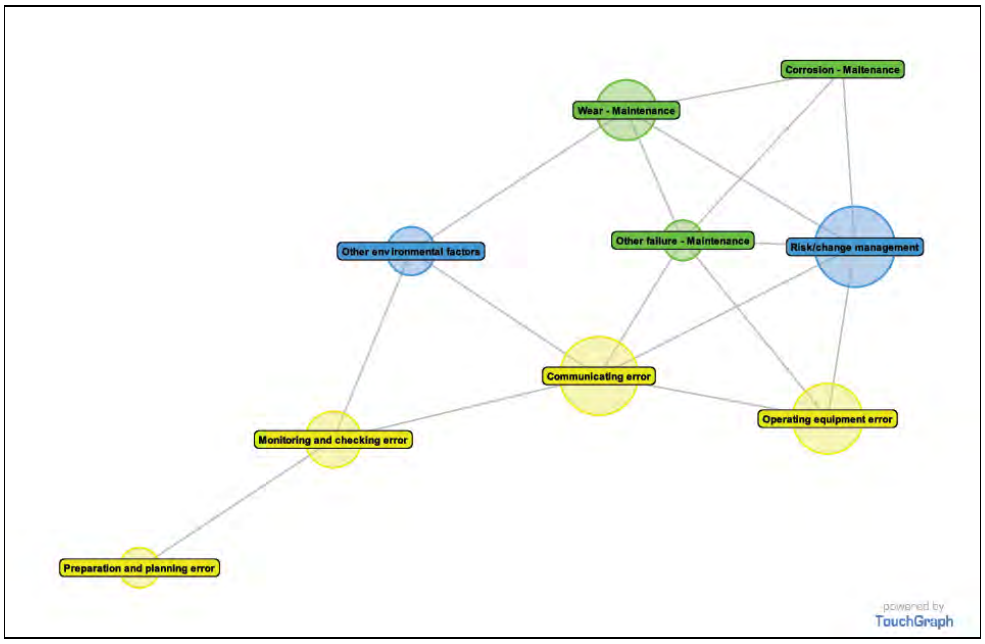

Corrosion - Maitenance
Wear - Maintenance
Other environmental factors
Other failure - Maintenance
Risk/change management
Communicating error
Operating equipment error
Monitoring and checking error
Preparation and planning error
powered by TouchGraph

Figure 3: Railway Yard Derailments Factors with Betweenness Centrality Measure > 1 Shown

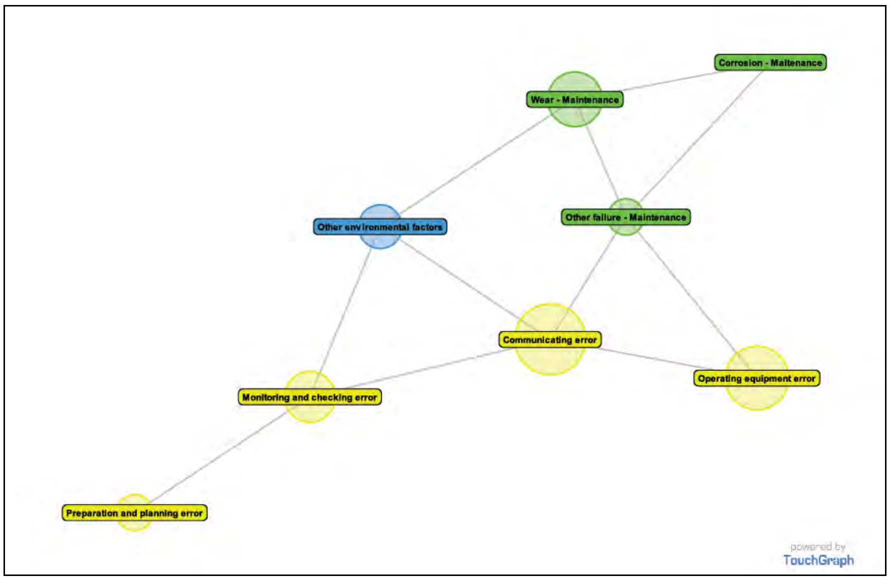

Corrosion - Maltenance
Wear - Maintenance
Other environmental factors
Other failure - Maintenance
Communicating error
Operating equipment error
Monitoring and checking error
Preparation and planning error
powered by TouchGraph

Figure 4: Railway Yard Derailments

Showing the model with the most central factor removed being *Risk/Change Management*

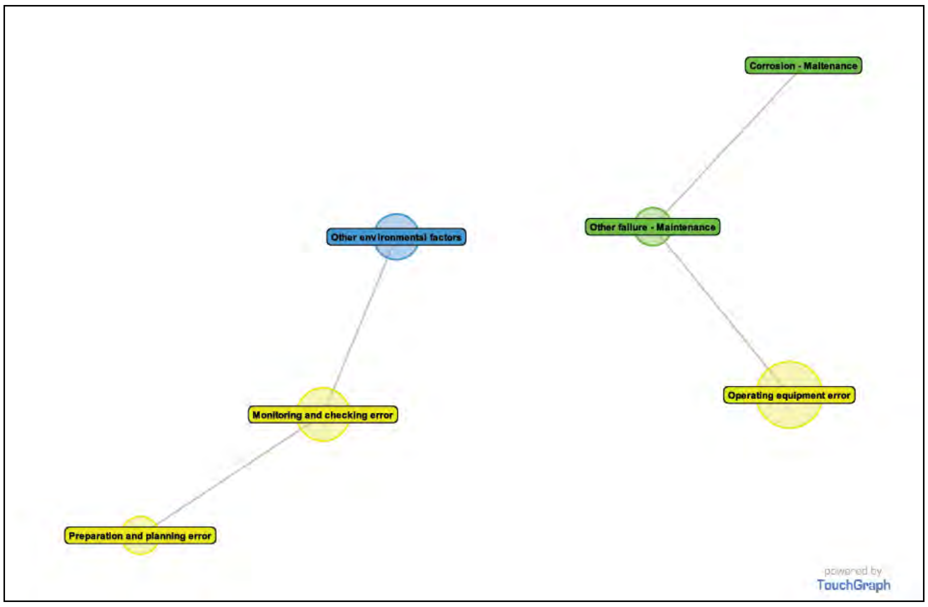

Corrosion - Maltenance
Other environmental factors
Other failure - Maintenance
Operating equipment error
Monitoring and checking error
Preparation and planning error
powered by TouchGraph

Figure 5: Railway Yard Derailments

Showing the model with the next highest Betweenness centrality measure factors removed

Being *Communicating Error* and *Wear Maintenance*

Model is now Broken Up

Statistical results are also available which are centred on network metrics in the form of centrality measures which determine the relative position of the contributing factors within the larger network as show in Table 1. Network analysts use centrality measures to determine the 'importance' or 'most central' factors in the network. In a typical social network problem, these central factors might be key people who are 'influencers', but in the present analyses instead represent the key contributing factors in types of railway safety occurrences.

Halo circle size around the factors indicate the strength of the Betweenness measure of centrality (a measure of how much removing a factor would disrupt the connections between one factor and another factor), i.e., those with the largest halos have the highest measure of betweenness centrality and therefore their removal from the network would disrupt the other connections. They act as a bridge between other network factors and are integral to maintaining the network structure. Betweenness centrality may be important in accident modelling, because theoretically the effective disruption of a network of contributing factors may prevent a future safety occurrence from manifesting. This premise lies at the heart of preventative risk management safety strategies which expect that the control or removal of identified risk factors will lead to the prevention of safety occurrences.

Figures 3 to 5 show some of the types of manipulations available when examining the model outputs from SNA. Figure 3 shows the first model reduced to show only those factors having a measure of Betweenness Centrality greater than one. Whilst figures 4 and 5 show the network model reduced to the point of 'breaking up' with the removal of the most central factors.

This examination is particularly interesting to the safety professional as it may enable the identification of those factors which, from a risk management control framework point of view, are removed or controlled could be seen as having an influence on the prevention of future safety occurrence of the same type.

Table 1 Show the Betweenness Centrality measure for Railway Yard Derailments for any factors which have a Betweenness Centrality measure of greater than 1.

Factor Name	**Betweenness Centrality**
Risk/change management	182.16
Communicating error	171.33
Operating equipment error	142.83
Wear - Maintenance	102.5
Monitoring and checking error	92
Other environmental factors	65
Other failure - Maintenance	48.33
Preparation and planning error	48
Corrosion - Maintenance	3.83

Table 1: Railway Yard Derailments Betweenness Centrality

Here the Local Condition & Organisational factor known as Risk/Change Management is indicated as having the highest Betweenness Centrality measure. Communicating Errors is a close second and Operating Equipment Errors comes third. Whilst Wear Maintenance comes fourth, interestingly its removal along with Communicating Errors breaks up the network as illustrated in Figure 5. Having both a visual model and centrality measures is proving valuable in knowing 'where' to focus effort in preventing future incidents.

8 Results

Figures 2 to 5 show typical SNA outputs for the contributing factors identified for one type of major rail safety occurrence: yard derailments. SNA provides information on both a visual level (an illustration of the accident model) and also allows statistical analysis through measures of centrality.

The resulting Accident Model for the network of contributing factors for Yard Derailments for the years 2006 to 2010 is shown in Figure 2. The visual network or accident model, along with the data results as shown in Table 1 indicate that the Local Condition & Organisational factor of *Risk/Change Management* has the highest measure of Betweenness Centrality followed by the Individual/Team factor of *Communication Error*, followed by the Individual/Team factor of *Operating equipment error*.

Figure 3 shows the accident model reduced to show only those factors which have a betweenness centrality measure of greater than zero. Those factors whose betweenness centrality measure is zero have little importance in the network, as their removal would have no affect on the other network factor connections. Filtering in SNA enables better understanding of the visual network where a large number of nodes is common and where the network can then appear very complex. By reducing the number of nodes (in this case factors) under review to only those with some centrality importance by intelligently filtering the network, the network is easily understood in terms of centrality measures (the most central contributing factors) and importance of the remaining factors within the network.

Figure 4 shows the network after the removal of *Risk/Change Management* as the highest ranked Betweenness Centrality factor. Figure 5 shows the network after the removal of two more factors, including the second highest ranked betweenness central factor of *Communication Error* and the fourth ranked Betweenness Centrality factor being *Wear – Maintenance,* and shows how the network is best disrupted. A disrupted network may be best thought of as a potential incident that never occurs (or is recorded) because of a lack co-occurring contributing factors to produce a valid safety concern – as represented in the existing dataset.

Whilst the centrality output data identifies that the third most Betweenness Centrality ranked factor was *Operating Equipment Error* the visual network provides a pattern that shows that the network can be broken up in three steps with the removal of the fourth factor rather than having to remove the third factor and then the fourth factor. This is one of the benefits of using this type of method for modelling safety occurrences, with the visual network showing the linkages and relationships amongst the factors giving a clearer understanding of the factor networks rather than just relying on data interpretations. Preventative safety efforts can then be correctly directed to those contributing factors which play a central role in the network and whose elimination would have the effect of disrupting the network of factors which hold the model (accident) together, allowing it to occur.

9 Conclusions

The use SNA to produce an Accident Model has been demonstrated specifically for one type of railway safety occurrence being Yard Derailments. The accident model allows for a determination of which factors are contributing the most to types of safety occurrences under investigation and the contribution that each factor has in the overall network. This methodology known as the Safe-NET method has been shown to result in both a visual accident model as well as being able to produce measures of centrality for each of the contributing factors displayed in the model.

This paper has described the use of Betweenness Centrality measures as one type of further analysis available to the researcher using Social Network Analysis. This method expands on previously limited sequential representations for accident analysis i.e. event trees which attempt to demonstrate some form of network connectivity but which are limited to the analysis of only one safety occurrence at a time and are not truly a network analysis method or analysis.

Traditionally safety analysts studying accident contributing factors have been limited to looking at contributing factors as mere number counts i.e. the number of times it did occur. This meant that these results were limited to only identifying those factors which had the highest number counts, usually across separate streams or groups such as local conditions & organisational factors, human error factors and technical factors. This has resulted in valuable information on the relationship amongst the factors being missed and the understanding of the role of the most central factors and their importance in the safety occurrence being previously unavailable without the use of SNA.

The resultant accident models and data outputs from SNA programs now enables complex socio-technical systems to be examined and understood in a new way. These complex systems can now be modelled in a much superior reflective and realistic way for the actual system under examination.

The removal of central contributing factors is able to be shown visually and this has perhaps the most interesting result for the OHS professional by indicating which factors should be focussed on under a risk management regime, in an attempt to prevent the incident from reoccurring in the future.

Whilst traditional number counts of contributing factors allowed some insight into which factors were important, the use of SNA now allows the interaction of factors to be understood at a deeper level and the focus of safety efforts in accident prevention to be focussed on the most important and central factors from a systems perspective. It helps us to potentially prevent the co-occurrence of several contributing factors that tend to repeatedly appear together in many recorded incident events.

The visualisation of the network metrics reveals important properties about the individual contributing factors in the network. This can allow management's attention to be put to work on disrupting those contributing factors, which break the set of relationships that seem to be present in typical and all too often reoccurring railway incident scenarios. In the examination of types of railway incidents over a five year period Safe-NET is able to show where maximum benefit can be obtained by directing preventative measures towards those factors which are having the most influence in an attempt to prevent their reoccurrence in the accident phenomenon.

The Safe-NET method has been shown to generate a model of multiple types of actual safety occurrences under investigation, rather than being a model which just directs an investigative method. This is in keeping with the recent call for modern day accident models to be representative of actual accident phenomenology.

Perhaps for the first time, through the use of the methodology demonstrated here, a resulting accident model can be truly representative of both the actual safety occurrences under investigation and representative of the complex socio-technical system in which it occurs.

10 References

Buzsáki, G 2006 *Rhythms of the Brain*, New York, Oxford University Press.

Goh, YM, Brown, H & Spickett, J 2010 'Applying systems thinking concepts in the analysis of major incidents and safety culture', *Safety Science,* vol 48, pp. 302-309.

Heinrich, HW 1931 *Industrial accident prevention: A scientific approach*, New York, McGraw-Hill.

Contributing Factors Framework 2012 Available from, www.onrsr.com.au/safety-improvement/contributing-factors-framework

Hansen DL, Shneiderman B & Smith, MA 2001 *Analyzing social media networks with NodeXL: insights from a connected world*, Boston, Elsevier Inc.

Newman, MEJ 2010 *Networks: An Introduction*, Oxford, Oxford University Press.

Roelen, ALC, Lin, PH, & Hale, AR 2010 'Accident models and organisational factors in air transport: The need for multi-method modles', *Safety Science,* vol 49, pp 5-10.

Senge, P 2006 *The Fifth Discipline - The Art & Practice of the Learning Organisation*, New South Wales, Random House Australia.

Sterman, JD 2000 *Business dynamics. Systems thinking and modelling for a complex world*. McGraw Hill, New York.

TouchGraph Navigator 2, 2013 available from: http://www.touchgraph.com/navigator

System Hazard Analysis of a Complex Socio-Technical System: The Functional Resonance Analysis Method in Hazard Identification

Brendon Frost[1] and John P.T. Mo
School of Aerospace, Mechanical and Manufacturing Engineering
Royal Melbourne Institute of Technology
GPO Box 2476, Melbourne 3001, Victoria

[1]brendon.frost@gmail.com

Abstract

The value of characterising systems in high-risk industries as complex and socio-technical systems is increasing. Using complex and socio-technical system view-points, the potential for unintended design risk factors to arise from complex and non-linear interactions can be highlighted, through focusing attention to the interactions between skilled operators, technology, and automation in geographically dispersed operations.

Currently there are methods for identifying and assuring the safety of interactions between and within systems but the modeling is incomplete. However, one potentially valid method is the Functional Resonance Analysis Method (FRAM). FRAM is a qualitative approach generating a functional (rather than structural) model of the relationships between sub/systems, and has the potential to produce inputs suitable for safety assurance and risk analysis methods.

This paper presents a methodology for incorporating a modified FRAM technique within a System Hazard Analysis (SHA). The application of the FRAM/SHA methodology in this case study is to explore the validity of this approach for assessing hazards arising from structural and process changes to the operational control center of an international airline, as a result of the introduction of a new software system into an existing suite of COTS software tools. This paper will explore the methodology in terms of requirements that include the creation of a new work team, the functional division of an existing operator role, and changes to system performance and safety critical processes.

Keywords: Socio-Technical system, Functional Resonance Analysis Method, FRAM, System Hazard Analysis, operational control center, risk analysis, COTS requirements.

1 Introduction

A significant challenge in designing and operating complex systems is the potential for unpredictable system behavior to ‘emerge’ from the complex (and often transient) interconnections that can arise under dynamic operating conditions.

Limited understanding of these factors can manifest as an undefined gap between the system as intended and the system as implemented/operated (Leveson, 2011a).This arises partly as a result of the limited guidance available to support the design of effective interactions between system elements (such as human operators and technology), particularly where they are separated by time and geography, and where system function is constrained by time, conditions of information uncertainty, and decentralised control mechanisms (Vicente, 1999).

In addition, the development of tools for analysing and modeling complex and non-linear system behavior (particularly under degraded performance modes), and the assessment of organisational, social, and human factor impacts, are in their relative infancy when compared to the traditional reliability driven approaches used across the system engineering life-cycle (Allenby & Kelly, 2001).

This paper describes a novel systems modeling technique, the Functional Resonance Analysis Method (FRAM), modified for use within the context of a System Hazard Analysis (SHA). FRAM has been used within the Cognitive Systems Engineering domain to model and assess the complex functional interactions between elements within socio-technical systems, but primarily for accident investigation purposes (Hollnagel, 2012). However, the technique potentially has greater utility than traditional hazard identification techniques (e.g. HAZOPs variants, FMEA/FMECA approaches, etc.) in supporting risk analysis through identifying hazards and defining the specific (and often transient) scenarios/conditions under which they arise.

The new modeling technique is illustrated by a case study of the Operations Control Centre (OCC) of a large international airline.

2 Complex System SHA

The identification of potential hazards in complex systems is fundamental to establishing safety and reliability across the design, production, and operational stages of the System Engineering (SE) lifecycle (Bahr, 1997). A complete and comprehensive picture of the hazards present or likely to exist in a system, and determination of the escalation of hazards to become mishap or accident scenarios, supports the early establishment of performance and safety requirements for design and operational objectives. This provides a more complete and comprehensive analysis to reduce the uncertainty of any subsequent risk quantification, while providing higher levels of safety assurance (Seligmann et al, 2012).

The SHA is a SE hazard analysis process applied during the design phases to assess the integration of designs. The SHA aims to identify hazards arising from the functional interfaces between subsystems, as well as the presence of latent design hazards with the potential to escalate into interrelated fault events (Ericson, 2005). Roland and Moriarty (1990) list the primary objectives of the SHA process as including consideration of:

- Compliance with specified safety criteria;
- Hazardous events, including failure or degradation of safety devices, controls, and safety constraints and functions;
- Degradation of the system's safety levels under normal or abnormal conditions;
- The impact of design or engineering changes, and;
- Human System Interfaces (HSI) including human performance errors and control functions.

The SHA process may draw from a broad range of established hazard identification techniques, but will typically commence with qualitative techniques to establish the causality of credible mishap/accident scenarios before proceeding to quantification techniques (NASA, 2011).

The foundation of hazard identification and analysis (HAZID), therefore, is identifying credible mishap and accident event scenarios. However, HAZID within safety critical systems is often challenging due to the stochastic effects of interactive and dynamic system complexity, and the presence of system intractability (or under-specification), as commonly found in complex and socio-technical systems (Hollnagel, 2012). Established risk analysis practice adopts the scenario driven approach to the systematic review of safety-critical systems, in order to identify potential mishaps and accidents (CCPS, 2008; Mannan, 2005).

The development of credible scenarios enables system complexity to be reduced to discrete 'snapshots' of the system under a range of conditions and in different system states, where they can be modeled as deterministic functional relationships between system elements (Bossel, 2007). A scenario consists of an expected situation/characteristic sequence or combination of events, and describes a generic situation that encompasses and relates a set of reasonably probable events/situations. Khan (2001) cautions against focusing on identification of the 'worst-case' scenarios through risk assessment activities, as this may unnecessarily restrict the scope and coverage of the scenario set identified: he suggests that it is preferable to focus efforts on identifying *"...credible accident[s] ... within the realm of possibility and likely to be severe enough to cause significant damage"*.

An effective risk analysis process for a complex system must therefore combine appropriate hazard identification and analysis techniques in order to generate as complete and comprehensive a set of mishap and accident scenarios as possible for subsequent quantification (Cameron & Raman, 2005). Siu (1994) particularly identifies dynamic system dependencies (e.g. common-cause initiators, functional coupling, and shared equipment/components) as requiring 'complexity decomposition' before the modeling approaches commonly used in risk analysis can provide valid scenario quantification. In addition, Leveson (2011a) suggests that the selection of HAZID techniques should consider the system's complexity and socio-technical characteristics, so that the HAZID process is able to describe system scenarios resulting from dependent incredible events, and/or transient system states.

System performance variability is a common feature of large-scale socio-technical systems, where demands arising from interaction with the external environment, social, organisational, and individual operator system factors within the system, must be met through trade-offs against the purpose and objectives of the system and within finite time and resource constraints. This leads to a situation where a complete description of the system of work (i.e. how work is to be accomplished) is intractable, or cannot be fully specified due to the effect of elaboration (the presence of significant detail), the rate of change (dynamic complexity), incompleteness of functional knowledge, and/or process heterogeneity and irregularity (Hollnagel, 2012). Performance variability therefore can be seen as a response to the presence of dynamic complexity in the system over time.

Socio-technical systems can be defined as having a human-intensive and organisation focused architecture, and are an increasingly common class of large-scale systems that feature a combination of technological systems (where hardware and software technology feature as significant elements within the system), human interfaces, and human-intensive organisational systems (Jackson, 2010).

Common to these complex and large-scale socio-technical systems are characteristic behaviors (Bossel, 2007). Consideration of these system socio-technical and complexity factors informs the decomposition of system structure and function as a basis for identifying how perturbations of, and interactions within, the system under study can propagate in undesirable ways (Dekker, Cilliers & Hofmeyer, 2011).

A number of approaches have been suggested for conducting HAZID processes while taking complexity and socio-technical factors into account: however, to date none have achieved widespread adoption outside of specific communities of practice. Most address these challenges through modification of existing techniques/approaches, but two techniques have been specifically developed to address this need:

- *System Theoretic Process Analysis* (STPA) (Leveson, 2011a, 2004); and,
- *Functional Resonance Analysis Method* (FRAM) (Hollnagel 2012, 2004).

FRAM was chosen as the basis for this research partly because of the body of literature available, but primarily because of it's potential for modeling graduated/degraded functional variability (Herrera & Woltjer, 2010). FRAM is most likely to be of value when applied from the detailed design phase of the Systems Engineering life cycle onwards, and following the availability of a detailed Concept of Operations (CONOPS). In this context FRAM could replace and/or complement established techniques such as HAZOPs, Functional Failure/Hazard Analysis, etc., and is likely to identify system hazards missed during the

Preliminary Hazard Identification (PHI) process.

3 FRAM in Context

The Functional Resonance Analysis Method (FRAM) is a qualitative analysis technique that supports the modeling of complexity and socio-technical factors, including the interfaces between adaptable human agents and technology, coupling and dependence effects, non-linear dependencies between sub-systems, and functional performance variability (Woltjer & Hollnagel, 2008b).

FRAM has previously been used to conduct qualitative system analyses: initially as a system accident investigation technique (Herrera & Woltjer, 2010), and more recently as a self-contained qualitative risk assessment method to inform design activities for large distributed systems (see: Belmonte et al, 2011; Herrera & Woltjer, 2010; Macchi et al, 2008; Woltjer & Hollnagel, 2008a; Woltjer & Hollnagel, 2008b). To date, FRAM analyses have been undertaken for air traffic management, rail transport, financial market, and nuclear waste transport systems.

FRAM theory contains different definitions of terms to those commonly used in HAZID activities:

- Functions are defined as representing the set of activities (the actual or likely work done, rather than an idealized work-as-imagined) required to produce an outcome or achieve sub/system objectives. More formally, the concept of a function is associated with activity intended to produce something of relevance to the system's objectives or change system state; the function's output describes a system condition or state (Hollnagel, 2012).
- Mishap/accident scenarios are seen as a product of uncontrolled hazards that emerge from performance variability and led to unintended or increased functional interactions and dependencies within a system, causing a sub-system/element event to cascade and resonate through the system (Hollnagel, 2004).
- Performance variability is considered to arise from the intractability of work management within complex systems, as agents independently trade-off efficiency against thoroughness (known as the Efficiency-Thoroughness-Trade-Off: ETTO) in achieving the purpose and objectives of the system (Hollnagel, 2009).
- Failure is defined differently to existing HAZID techniques: "*the temporary or permanent loss of a system's ability to anticipate risks and make proactive approximate adjustments to understand and adjust to the current conditions (resources, demands, conflicts, interruptions, underspecified work requirements)*" (Hollnagel, 2013b).

FRAM supports a systemic decomposition methodology, with analysis of a unique FRAM model (of a specific system) describing the functionality needed to meet the system's objectives and the range of functional variation that supports the achievement of these objectives (Hollnagel, 2013a). Through characterizing the variability of these functions, specific *instantiations* (or snapshots) of the system under defined situations and conditions can be determined to identify how interactions and relationships within the system (and with other systems) reconfigure, leading to undesirable functional variability and resonance within the system (Hollnagel, 2012).

This is where FRAM differs in concept from established HAZID techniques, in that it focuses on determining the likelihood of functional variability rather than the probability of malfunction or failure (in the traditional use of the term as the loss of function). However, in applications the FRAM approach remains compatible with established HAZID techniques used in many industries, in that the technique guides the decomposition and/or analysis of the system under study in order to identify plausible scenarios that form the basis for subsequent risk assessment activities.

While the majority of the FRAM process involves the development of the model in a table format, FRAM models and instantiations can also be represented in a graphical form using a modular 'hexagon' representation of each function and its six defining characteristics. An example of a FRAM module graphic is shown in figure 1.

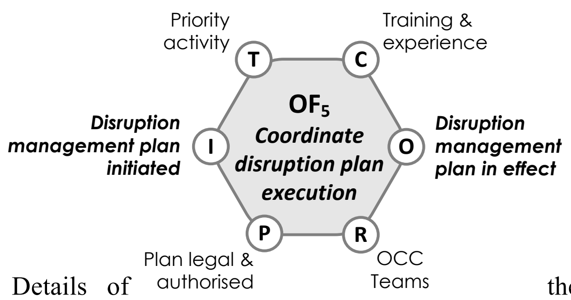

Priority activity
Training & experience
T
C
OF5
Coordinate disruption plan execution
Disruption management plan initiated
I
O
Disruption management plan in effect
P
R
Plan legal & authorised
OCC Teams

Figure 1: A graphic representation of a FRAM module

Details of the standard approach for applying FRAM can be found in Hollnagel (2012). The FRAM/SHA method used in this research differs from that published (Hollnagel, 2012; Hollnagel, 2004) as follows:

- Once the model had been constructed, a small, representative group of system experts were consulted to both refine/confirm/agree that the model was representative of the intended functional system, and to provide advice on the nature of credible functional 'perturbations' likely in the system.
- The baseline FRAM model was reviewed by an expert group of system experts in a facilitated HAZID workshop following a guideword process, in order to identify hazards, their escalation paths to incidents/accidents (represented by sets of discrete scenarios that formed the primary products/outputs from the HAZID process), and the barriers in place to prevent this.
- The validated model and credible system perturbations were then used to develop 'instantiations' of the model, representing the revised couplings between functions and any impacts on function performance/efficacy under defined conditions.

- The HAZID workshop outputs and three FRAM instantiations were then used as the basis for developing risk models.

The adoption of the guideword concept from (amongst others) the popular HAZOPs technique was used to improve the efficiency of the HAZID workshop process through consistent prompting and guidance of system experts. The function aspects defined in the FRAM (i.e. input, time, preconditions, resources, controls, output) can be considered equivalent to the 'keywords' concept from HAZOPs, which act to focus attention on the parameters of interest. Specific guidewords were developed to facilitate the FRAM process and ensure consideration of two different aspects of function variability:

- Factors affecting the variability of the function itself due to the operation of the function (i.e. internal influences on variability), through determining different aspects of the output variability,
  and;
- Factors affecting inputs to the function that affect the Input, Time, Control, Resource, or Precondition characteristics of the function (i.e. external influences on variability).

Consequently, this modified FRAM method used two separate sets of guidewords, applied separately in two sequential 'passes' across the baseline FRAM model, for the system experts to identify hazards arising from interactivity under defined conditions, and the effect of potential variability on function coupling and performance:

Pass 1: Determination of function variation where the input, time, control, precondition, resource is *Early*, *Delayed*, *Absent*, *Wrong Rate*, *Under specified* (insufficiently constrained), *Over-specified* (too constrained).

Pass 2: Determination of variation where the function is impacted by the specific conditions or socio-technical and individual factors of:

- *Time pressures/constraints/duration*
- *Information certainty/sufficiency*
- *Quality of human-technology interfaces*
- *Quality of communications*
- *Procedures and authorisations*
- *Competence and preparation (e.g. training)*
- *Goal objectives/conflicts*
- *Circadian and stress factors*
- *Organisational influences and support*

Despite the complexity of the system problem studied in this research, the available access to system experts (both in terms of total contact time and the number of individuals) was limited. As a result, the development of the FRAM process used in this paper was influenced by this need to minimize the total system expert resource required, and the time that they were required. In practice this led to a number of process efficiencies:

- The model validation step (and identification of system perturbations) could be conducted quickly (around 1 hour) with a small number of system experts (2 individuals were sufficient for this analysis);
- The HAZID workshop could be constrained to a session of just a few hours (2 hours were all that was available for this research), as a concise number of representative instantiations (3 were sufficient to explain the outputs of the process for this research) of the FRAM model were developed following the workshop and used to precisely focus and contain the HAZID process.
- The validation of hazard scenarios used graphic FRAM model representations, as a convenient way to represent process differences due to functional interaction and coupling under different scenarios, in a way that allowed domain and specialist experts a common terminology or representation upon which to agree likely HAZID outcomes (Nemeth & Bartha, 2009).

It can be seen that the involvement of system experts is required in order to ensure model validity, improve the quality of the analysis and utility of the process products, and to make explicit the objectives of each process step for all involved.

4 The Airline Operations Control System

Operations in a modern airline environment are complex, requiring the coordination of a number of different business units to ensure the availability and operation of aircraft to meet strategic objectives within market commitments and opportunities. These services are coordinated and directed to manage operational disruptions due to weather, technical failures, air traffic and airspace/aerodrome management, with oversight by the Operations Control Centre (OCC). The OCC typically monitors all aspects of the airline's daily operations, with key accountabilities including:

- Maintaining the integrity of the planned operational service schedule.
- Responding to any known and controllable risks that could cause variation to the planned operational service schedule.
- Managing variation to the planned service schedule, including developing, implementing, monitoring, and managing plans to minimise or recover from unplanned operational disruptions.

A graphic representation of the proposed OCC system, interfaces, functional flows, and COTS software is provided in figure 2. A baseline FRAM model of the proposed OCC was developed and validated as described in the previous section, and is presented in figure 3. The model shows the twenty-one key OCC functions required for disruption management activities, in addition to one external input and two external output functions.

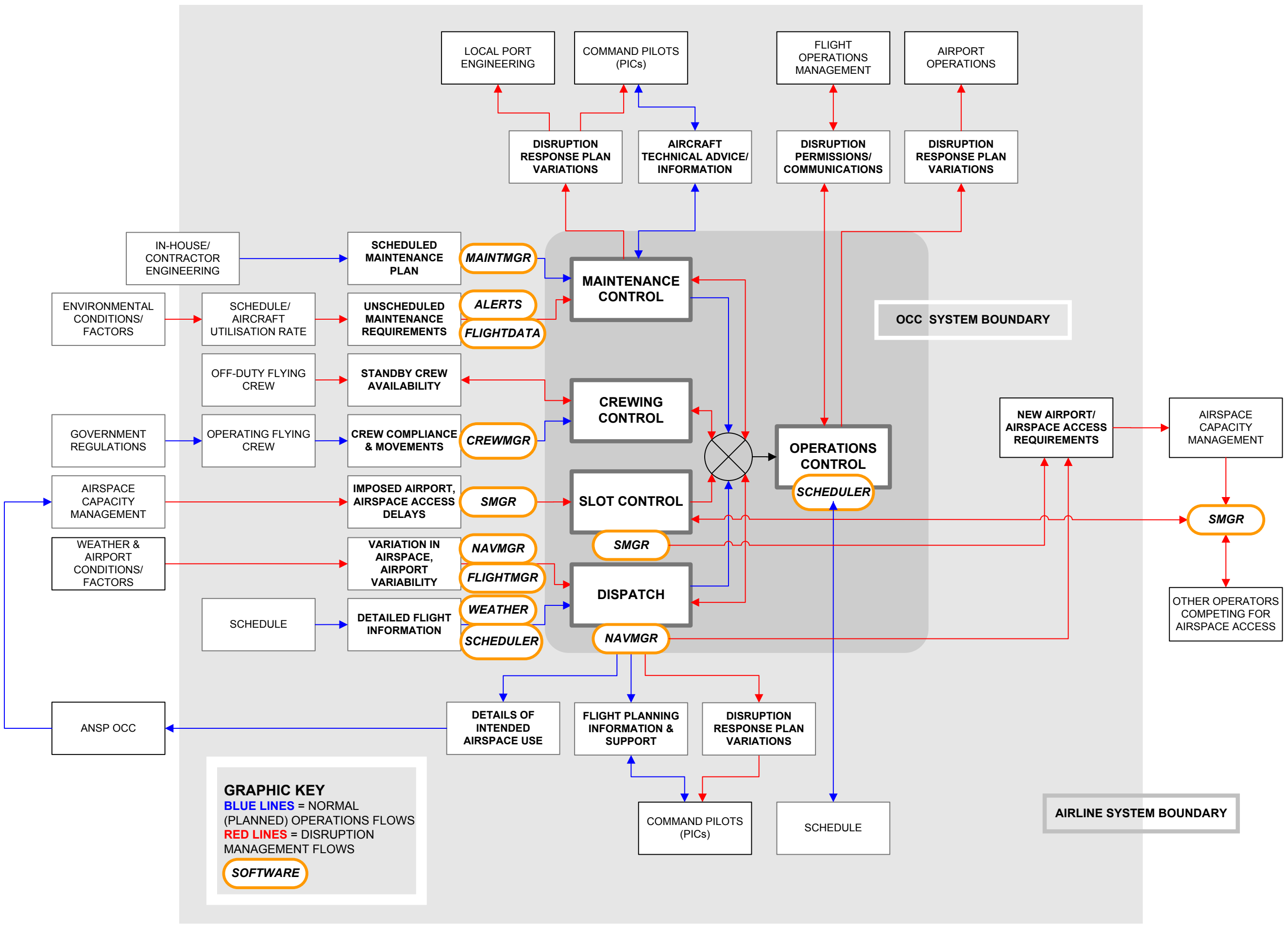

LOCAL PORT ENGINEERING
COMMAND PILOTS (PICs)
FLIGHT OPERATIONS MANAGEMENT
AIRPORT OPERATIONS
DISRUPTION RESPONSE PLAN VARIATIONS
AIRCRAFT TECHNICAL ADVICE/ INFORMATION
DISRUPTION PERMISSIONS/ COMMUNICATIONS
DISRUPTION RESPONSE PLAN VARIATIONS
IN-HOUSE/ CONTRACTOR ENGINEERING
SCHEDULED MAINTENANCE PLAN
MAINTMGR
MAINTENANCE CONTROL
ENVIRONMENTAL CONDITIONS/ FACTORS
SCHEDULE/ AIRCRAFT UTILISATION RATE
UNSCHEDULED MAINTENANCE REQUIREMENTS
ALERTS
FLIGHTDATA
OCC SYSTEM BOUNDARY
OFF-DUTY FLYING CREW
STANDBY CREW AVAILABILITY
CREWING CONTROL
GOVERNMENT REGULATIONS
OPERATING FLYING CREW
CREW COMPLIANCE & MOVEMENTS
CREWMGR
OPERATIONS CONTROL
SCHEDULER
NEW AIRPORT/ AIRSPACE ACCESS REQUIREMENTS
AIRSPACE CAPACITY MANAGEMENT
AIRSPACE CAPACITY MANAGEMENT
IMPOSED AIRPORT, AIRSPACE ACCESS DELAYS
SMGR
SLOT CONTROL
SMGR
SMGR
WEATHER & AIRPORT CONDITIONS/ FACTORS
VARIATION IN AIRSPACE, AIRPORT VARIABILITY
NAVMGR
FLIGHTMGR
DISPATCH
OTHER OPERATORS COMPETING FOR AIRSPACE ACCESS
SCHEDULE
DETAILED FLIGHT INFORMATION
WEATHER
SCHEDULER
NAVMGR
ANSP OCC
DETAILS OF INTENDED AIRSPACE USE
FLIGHT PLANNING INFORMATION & SUPPORT
DISRUPTION RESPONSE PLAN VARIATIONS
GRAPHIC KEY
BLUE LINES = NORMAL (PLANNED) OPERATIONS FLOWS
RED LINES = DISRUPTION MANAGEMENT FLOWS
SOFTWARE
COMMAND PILOTS (PICs)
SCHEDULE
AIRLINE SYSTEM BOUNDARY

Figure 2: A systems representation of the proposed OCC System

4.1 Baseline FRAM Model Analysis

Key/notable features in the baseline FRAM model that are highlighted include:

- The OCC can only react to, not anticipate, disruption given the functional arrangement of the system and design of sub-system activities and support software.
- The role of Operations Control (OC) as the central control 'hub' for disruption management activities, previously suggested through the system description and decomposition processes, was confirmed. In addition, the baseline FRAM model provided additional insights into the type and nature of functional interactions within the OCC and with external systems and the environment, particularly:
  - The external disruption input originating with the release of the SMGR delay schedule.
  - The downstream flow of system outputs (information, instructions) to local operational elements tasked with carrying out disruption management tasks. The model disputes the assumption that early consultation and interaction with local operational elements, intended to occur at OC functions, are sufficient and accurate, and suggests that new feedback loops are required (other than the existing software systems representations) to confirm the disruption management plan has been executed in the local environment.

The assumed nature and type of interactions between OCC sub-systems was explored and some previously unappreciated interactions uncovered, including:

- All OCC areas have goal conflicts with each other, apply different trade-off parameters, and have overlapping objectives to be satisfied when responding to disruption events.
- The concentration of accountability with the OC for making sense of operational information and reducing information uncertainty, manage risk, and lead disruption management planning and responses, was noted. The limited accountability of OCC sub-systems for processing and integrating disruption indicators was highlighted.
- The reliance of OC on OCC sub-systems to monitor and advise of disruption conditions and events as promptly as possible.
- The nature of connections between existing OCC sub-systems (i.e. excluding the proposed Slot Control function) are predominantly 'green', reflecting normal and direct connections/coupling between OCC sub-systems including Operations Control.

Analysis of the model suggested a number of new couplings and interactions between the OCC sub-systems due to the introduction of the proposed Slot Control (SC) function:

- Tighter coupling appeared likely between OC and the proposed SC sub-system than seen with other OCC sub-systems. This suggested a different dynamic and raised the potential risk of uncoordinated OCC activities as a result of a lack of clarity as to which of OC or SC would 'lead or follow' under different conditions – this was in contrast to the clear lead accountability that OC had in relation to existing OCC sub-systems.
- Increased complexity and coupling of OC functions, particularly as a result of dependence on SC functions. This had two main effects to increase complexity: a) an increase in the number of connections, particularly around the primary disruption management response functions, and; b) the nature of the new connections were predominantly 'blue', reflecting an increase in the indirect influence of SC functions on OC functions through the introduction of constraints and dependence on SC.
- Disruption in the OC disruption sequence, particularly with respect to new connections/coupling from SC functions that bypass the OC functional sequence and that have the potential to create uncoordinated activities between the two sub-systems.
- Finally, functions for SC (Monitor and identify slot non-compliance) and OC (Identify aircraft movement variations) are, from an OCC system perspective, essentially the same activities. SC and OC undertake this activity independently with different objectives, using the same data source (i.e. indicators of push-back at origin, takeoff, touch-down, and parking at destination), but interfacing with different interface representations on different support software. This has the potential to drive different views of a disruption event and the required response activities.

4.2 Derivation of Hazard Scenarios

Once the FRAM baseline model had been validated through System Expert workshop/s, a further series of two workshops employed the keyword/guideword approach to explore perturbations to the model in response to variations in operational conditions, excessive demand on the system, or performance failures within the different OCC sub-systems. The products of this process were:

- Defined sets of system conditions under which the OCC would functionally change, leading to changes in the functional arrangement, different interactions and coupling between OCC sub-systems, and in some cases the redundancy or obsolescence of OCC functions present in ideal conditions.
- Specific FRAM instantiation models of the different functional systems emerging from changes in the defined system conditions, with each instantiation model providing the basis for the development of specific HAZID scenarios.
- Specific HAZID scenarios suitable for formal risk modeling (e.g. full quantification through Probabilistic/Quantitative Risk Assessment, or semi-quantitative techniques such as Bow-tie Analysis, Safety Barrier Diagrams, etc.).

As a result of this process three instantiation models were identified through the FRAM process, linking together through a single Top Event, specifically:

An underspecified disruption event occurs in local operations leading to a time loss of >15 minutes, where: The disruption event information is incomplete, uncertain, or subject to time constraints.

The three representative instantiation models developed were SCF-1 (Slot Control Failure), HSI-1 (Human-System Interface), and DRM-1 (Disruption Management). Two of the instantiation models (SCF-1 and HSI-1) describe scenarios that lead to the defined Top Event – i.e. they identify possible causative pathways leading to the Top Event), whereas the third instantiation model (DRM-1) describes the scenarios and outcome events following the Top Event. In these FRAM models the common location for this Top Event is coincident with the primary OC function defined in the model.

4.2.1 Analysis of FRAM Scenario SCF-1

The Slot Control Failure instantiation model (SCF-1) describes the consequences of a situation where the proposed SC function fails to perform to the required standard, and the OCC undergoes a sudden reconfiguration with SC activities being commandeered by OC. The degraded SC sub-system performance is most likely after the initial implementation of the new OCC configuration, under conditions of high workload (such as during peak airspace usage times, in the lead-up to curfews at specific airports, etc.), and likely to be exacerbated by the relative inexperience of the Slot Control Operators during early operations with the new system. Analysis of the SCF-1 instantiation model suggests that:

- The transfer of functional activity to OC is likely to increase the volume of OC work tasks, and a larger information set will need to be tracked and integrated to produce an adequate description of the disruption event – this will introduce time delays into the disruption response process.
- OC officers are unlikely to have the training and competency with SMGR to enable the same ease of data identification and integration as SC operators, taking more time and with a greater risk of missing or miss-interpreting critical information.
- While sufficient controls exist with the airline and airspace systems to capture the effect of any errors and mitigate safety hazards/risks, the introduced time delays impact commercial performance by reducing the availability of suitable recovery options following a disruption event, some functions will be performed in a 'degraded mode', and the optimisation function in the model will be deactivated during disruption recovery.

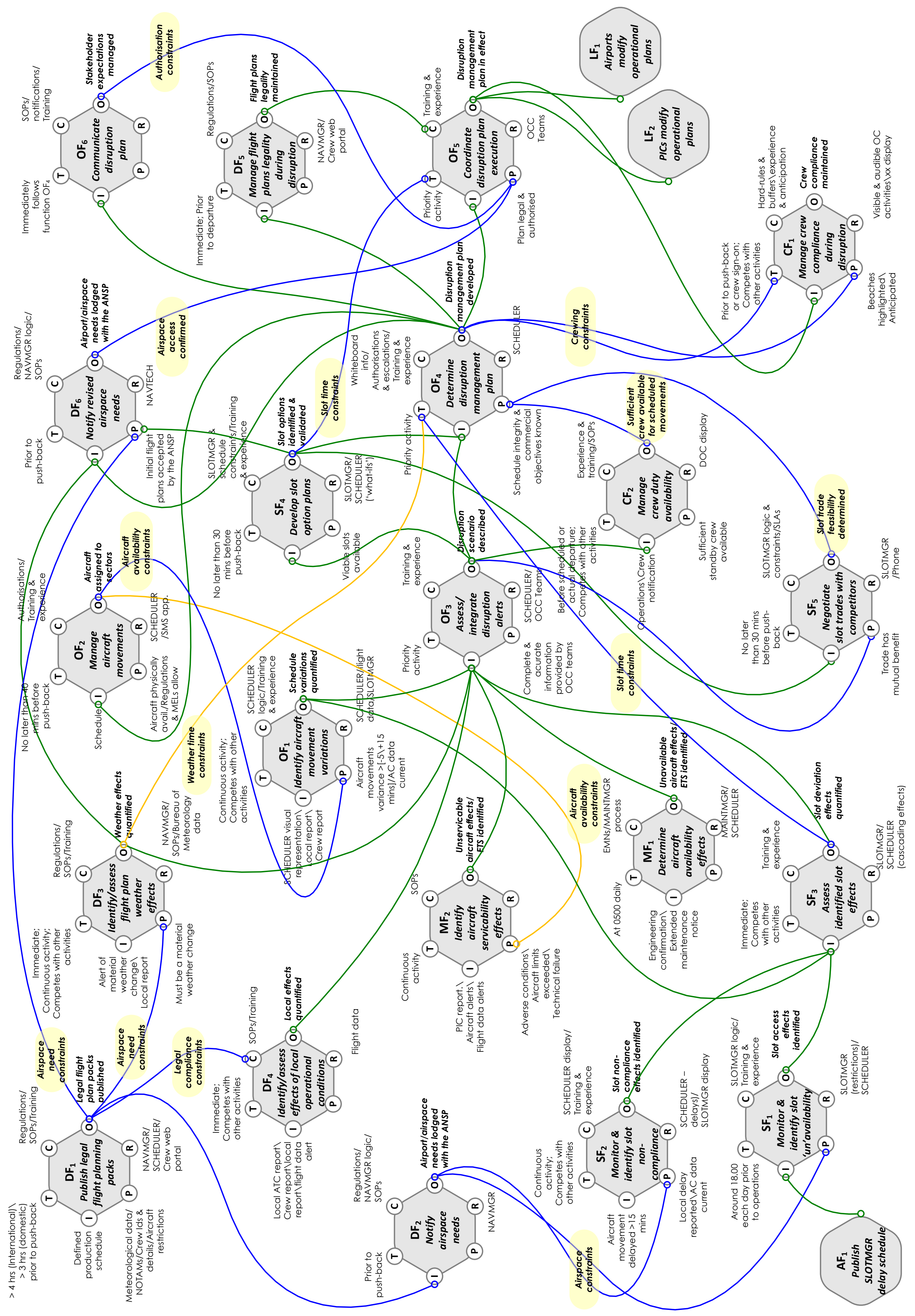

DF1 Publish legal flight planning packs
DF2 Notify airspace needs
DF3 Identify/assess flight plan weather effects
DF4 Identify/assess effects of local operational conditions
DF5 Manage flight plans legality during disruption
DF6 Notify revised airspace needs
OF1 Identify aircraft movement variations
OF2 Manage aircraft movements
OF3 Assess/integrate disruption alerts
OF4 Determine disruption management plan
OF5 Coordinate disruption plan execution
OF6 Communicate disruption plan
SF1 Monitor & identify slot 'un'availability
SF2 Monitor & identify slot non-compliance
SF3 Assess identified slot effects
SF4 Develop slot option plans
SF5 Negotiate slot trades with competitors
CF1 Manage crew compliance during disruption
CF2 Manage crew duty availability
MF1 Determine aircraft availability effects
MF2 Identify aircraft servicability effects
AF1 Publish SLOTMGR delay schedule
LF1 Airports modify operational plans
LF2 PICs modify operational plans
Stakeholder expectations managed
Authorisation constraints
Flight plans legality maintained
Disruption management plan in effect
Disruption management plan developed
Airport/airspace needs lodged with the ANSP
Airspace access confirmed
Slot options identified & validated
Slot time constraints
Crewing constraints
Sufficient crew available for scheduled movements
Slot trade feasibility determined
Disruption scenario described
Aircraft assigned to sectors
Aircraft availability constraints
Schedule variations quantified
Weather time constraints
Weather effects quantified
Unavailable aircraft effects/ETS identified
Unservicable aircraft effects/ETS identified
Slot deviation effects quantified
Legal flight plan packs published
Airspace need constraints
Legal compliance constraints
Local effects quantified
Slot non-compliance effects identified
Slot access effects identified
Airspace constraints
Crew compliance maintained

Figure 3: FRAM baseline model of the proposed OCC System

- SC shifts to a 'response on demand' mode solely triggered by OC requests; monitoring of aircraft movement is constrained, and subsequently becomes less capable of anticipating and projecting the likely future consequences/impacts of slot options.

In summary, the SCF-1 model proposes a situation where changes in system configuration, and an increase in coupling between SC and OC functions, lead to commercial impacts and reduced system performance.

4.2.2 Analysis of FRAM Scenario HSI-1

The Human-System Interface instantiation model (HSI-1) describes numerous Human-System and Human-Computer Interface issues, including a number of factors directly impacting on the accurate description of disruption events and the timely monitoring of disruption management plan implementation/execution. These factors included underspecified software data, obsolete information representations, information supply delays, support software limitations with information presentation, and limited support for 'what-if' modeling of possible scenarios.

These effects are present to varying degrees under all conditions, but are exacerbated and the Top Event identified previously more likely under conditions where local reporting is delayed or not available to reduce information uncertainty, or where time pressures increase the likelihood that OCC operators will fail to identify or misinterpret critical information, or where they incorrectly integrate information.

Analysis of the HSI-1 instantiation model suggests that there are a number of functional connection/coupling changes possible, all independent and that therefore could occur simultaneously under credible scenarios:

- The SMGR input function provides processed information that is conservative and restricts OCC activities. However, airspace access is subject to sudden capacity changes due to shifts in environmental conditions, resulting in a sudden state transition within (geographical sections of) the Australian airspace system; The type and timing of data delivery through SMGR is delayed and changes without warning, such that the OCC is unlikely to be able to take advantage of sudden increases in airspace system capacity.
- The impact of information incompleteness, uncertainty, and access delay, leads to a situation where OC becomes more reliant on the Dispatch and SC functions, resulting in tighter coupling between key OCC functions.

In addition, there are a number of possible effects likely to eventuate under enabling conditions:

- SMGR does not use absolute 'real-time' data, and the obsolescence of the data presented needs to be judged by the operator. This slows decision-making and forces a conservative approach that requires information uncertainty to be resolved to reduce disruption response risk, but incurs time and opportunity cost.
- The main data set used by other scheduling and planning systems within the OCC is underspecified, as it does not provide continuous data to track progress towards milestone events, nor for critical unanticipated failure events (such as an aborted taxi maneuver and return to the terminal).
- Finally, there are only limited links between some of the support software, leaving the sub-system operators to largely act as the link to process the different representations, cross-check the processed data, and integrate the information to support planning and decision making activities.

This analysis has shown that there are HSI features present in the OCC system and the different support software used by each OCC sub-system that lead to inefficiencies and delays in the identification of, and access to, critical information: the consistency of information delivery varies, and the completeness and certainty of information received is also variable. This serves to force the OCC in general, and the OC sub-system in particular, to rely more on local reports to adequately describe a disruption event, in addition to increasing communication and event validation workload.

Finally, these aspects of the HSI-1 instantiation model highlight the conditions that arise under a specific (and likely) set of operating conditions whereby the previously identified Top Event can occur. However, they can also contribute to the set of conditions associated with the situation identified in the previous SCF-1 instantiation model, which highlights a contradiction: OC increasing coupling with, and dependence on, SC immediately prior to a sudden system state transition where dependence is reduced and the actual coupling interaction altered.

4.2.3 Analysis of FRAM Scenario DRM-1

The Disruption Management instantiation model (DRM-1) describes a situation where the OC sub-system responds to a disruption event without an accurate description of the event, and directs local activities that are inconsistent with the actual operational requirements. This is most likely to occur in the situation described by the SCF-1 instantiation model and is strongly influenced by the presence of the conditions described in HSI-1 – particularly when coupled with conditions of high workload (such as during peak airspace usage times, in the lead-up to curfews at specific airports, etc.), and time pressures. Analysis of the DRM-1 instantiation model suggests that:

- There are significant increases in downstream coupling between functions, and a corresponding increase in system complexity: Existing connections to the supporting sub-system functions are significantly impacted or degraded, as a result of enacting incorrect instructions and undertaking activities that will be inadequate or ineffective in responding to the actual disruption event at the local operational level;
- The 'downstream' functions acting in local operations now play a significant role in quality checking the adequacy of the disruption response instructions for addressing the disruption event.

As a result a feedback loop is created between local (airport/aircraft) functions and key OCC functions: however, this feedback may be subject to a significant time delay, and reduce the opportunity for OC to recover an unplanned event without incurring increased commercial costs.
- Also of note is the fact that some of the local operational functions now become temporary functional controls for managing flight plan legality and crew duty limit compliance instead of the designated OCC functions.

The effect of unresolved information uncertainty, incompleteness, or delay on the OCC system is dramatic: decision-making and response to disruption events are significantly delayed or incorrect. The effect of implementing an erroneous disruption management plan is to shift the responsibility for identifying and preventing the error is transferred outside of the OCC system, with local airport and pilot operations and the local Air Traffic Controller acting as the barriers that prevent inappropriate operational activities. This creates a new and significant feedback loop, but one that carries a significant time penalty and associated commercial cost.

5 Conclusion

This paper has presented a novel methodology for conducting a SHA in a complex and safety-critical system, for the case of an airline OCC. FRAM analysis of changes to the OCC of an international large airline confirmed the presence of complexity and socio-technical system features.

Notable requirements identified emerged from the analysis of interactive, dynamic, and decompositional complexity, and highlighted potential system performance improvement options through consideration of:

- The degree to which the existing software systems supported operator activities and OCC functions, and the impact of introducing new COTS software into an existing software suite.
- Information/data flows and currency, with particular attention to the efficacy of feedback loops.
- Coordination between OCC sub-systems.

The use of this modified FRAM technique within a SHA context was useful for hazard identification purposes, and potential hazards were identified at the functional interfaces between sub-systems and systems outside of the boundary of the system under study, and in relation to latent functional design hazards. In particular, it enabled system-level mishap scenarios to be identified, within the context of relationships between structural elements and the functional capability required to meet the system purpose and objectives.

With the increasing emergence of large scale and complex systems, including those that evolve independently of a central organizing architecture, the importance of techniques such as FRAM that allow the exploration of system behavioral and complexity effects will become increasingly critical to architecting systems that are safe by design.

6 References

Allenby, K., Kelly, T., (2001): Deriving safety requirements using scenarios. *Proc. Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Canada, 27 Aug 2001 - 31 Aug 2001, pp.228-235

Bahr, N.J. (1997): *System Safety Engineering and Risk Assessment: A Practical Approach*. New York, Taylor & Francis.

Belmonte, F., Schön, W., Heurley, L., and Capel, R. (2011): Interdisciplinary safety analysis of complex socio-technological systems based on the Functional Resonance Accident Model: an application to railway traffic supervision. *Reliability Engineering and System Safety* **96**: 237-249.

Bossel, H. (2007): *Systems and Models: Complexity, Dynamics, Evolution, Sustainability*. Norderstedt, Books on Demand GmbH.

Cameron, I., Raman, R., (2005): *Process Systems Risk Management*. Process Engineering Series, San Diego, Elsevier Inc.

CCPS, (2008): *Guidelines for Hazard Evaluation Procedures*. 3rd edition, Hoboken, American Institute of Chemical Engineers.

Dekker, S., Cilliers, P., Hofmeyr, J.-H. (2011): The complexity of failure: Implications of complexity theory for safety investigations. *Safety Science* (**49**): 939-945.

Ericson, C.A., (2005): *Hazard Analysis Techniques for System Safety*. Hoboken, John Wiley & Sons, Inc.

Herrera, I.A., Woltjer, R. (2010): Comparing a multi-linear (STEP) and systemic (FRAM) method for accident analysis. *Reliability Engineering and System Safety* (**95**): 1269-1275.

Hollnagel, E., (2004): *Barriers and Accident Prevention*. Surrey, Ashgate Publishing Limited.

Hollnagel, E., (2009): *The ETTO Principle: Efficiency-Thoroughness Trade-Off, Why Things That Go Right Sometimes Go Wrong*. Surrey, Ashgate Publishing Limited.

Hollnagel, E., (2012): *FRAM: the Functional Resonance Analysis Method: Modeling Complex Socio-Technical Systems*. Surrey, Ashgate Publishing Limited.

Hollnagel, E., (2013a): *Publications*. http://www.functionalresonance.com/FRAM_Publications.html. Accessed 12 Jun 2013.

Hollnagel, E., (2013b): *Introduction to FRAM: The Functional Resonance Analysis Method*. http://www.functionalresonance.com/FRAM-2_introduction_to_FRAM.pdf. Accessed 15 Jul 2013.

Jackson, S., (2010): *Architecting Resilient Systems: Accident Avoidance and Survival and Recovery from Disruptions*. Hoboken, John Wiley and Sons, Inc.

Khan, F.I. (2001): Use Maximum-Credible Accident Scenarios for Realistic and Reliable Risk Assessment. *Chemical Engineering Progress* **97**(11): 56-64.

Leveson, N.G., (1995): *Safeware: System Safety and Computers*. New York, Addison-Wesley publishing Company.

Leveson, N.G., (2011a): *Engineering a Safer World: Systems Thinking Applied to Safety*. Cambridge, The MIT Press.

Leveson, N.G. (2011b): Applying systems thinking to analyze and learn from events. *Safety Science* (**49**): 53-64.

Macchi, L., Hollnagel, E., and Leonhardt, J. (2008): A systemic approach to HRA: A FRAM modeling of Control Overflight activity. *4th Eurocontrol Annual Safety R&D Seminar*, Southampton, UK.

Mannan, S., (2005): *Lee's Loss Prevention in the Process Industries: Hazard Identification, Assessment and Control.* 3rd edition, Burlington, Elsevier Butterworth-Heinemann.

Modarres, S.W., Cheon, S.W. (1999): Function-centered modeling of engineering systems using the goal tree-success tree technique and functional primitives. *Reliability Engineering and System Safety* (**64**): 181-200.

NASA, (2011): *Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners*. NASA/SP-2011-3421, 2nd edition, Washington, NASA.

Nemeth, E., Bartha, T. (2009): Formal Verification of Safety Functions by Reinterpretation of Functional Block Based Specifications. In *FMICS 2008, LNCS 5596*. 199-214. Cofer, D., Fantechi, A. (eds.).

Nouvel, D., Travadel, S., and Hollnagel, E. (2007): Introduction of the Concept of Functional Resonance in the Analysis of a Near-Accident in Aviation. *33rd ESReDA Seminar: Future challenges of accident investigation*, Ispra, Italy.

Rasmussen B., Petersen, K.E. (1999): Plant functional modeling as a basis for assessing the impact of management on plant safety. *Reliability Engineering and System Safety* (**61**): 201-207.

Roland, H.E., Moriarty, B., (1990): *System Safety Engineering and Management*. 2nd edition, New York, John Wiley & Sons, Inc.

Seligmann, B.J., Nemeth, E., Hangos, K.M., Cameron, I.T. (2012): A blended hazard identification methodology to support process diagnosis. *Journal of Loss Prevention in the Process Industries* (**25**): 746-759.

Sui, N. (1994): Risk assessment for dynamic systems: An overview. *Reliability Engineering and System Safety* (**43**): 43-73.

Vicente, K.J., (1999): *Cognitive Work Analysis: Towards Safe, Productive, and Healthy Computer-Based Work*. Mahwah, Lawrence Erlbaum Associates.

Woltjer, R., Hollnagel, E. (2008a): Modeling and evaluation of air traffic management automation using the functional resonance accident model (FRAM). *8th International Symposium of the Australian Aviation Psychology Association*, Sydney, Australia.

Woltjer, R., Hollnagel, E. (2008b): Functional modeling for risk assessment of automation in a changing air traffic management environment. *4th International Conference Working on Safety*, Crete, Greece.

Fail Safe System and Software Architectures

Iain Henderson
HIMA Australia Pty Ltd
Level 3, 533 Hay Street
Perth WA 6000, Western Australia

ihenderson@hima.com.au

Abstract

Evidence supports the theory that the main source of software failures is due to systematic error during system and software requirements specification. Considerable effort is spent by Safety Engineers and the traditional system development teams in ensuring the development processes and techniques are of sufficient rigour to reduce the probability of systematic error. However, this will not always lead to the development of a robust system. To develop a system that can safely respond to erroneous or invalid system states requires a methodical and criteria based assessment approach.

Traditional assessment techniques such as Software Failure/Error Analysis and Software HAZOP can be adapted to assess the implications of implementing the functionality described in specifications and supporting design documentation. These adapted techniques can be applied to assess the system at three levels; one at the System Requirement level; one at the Software Architecture level, and one at the Software Detailed Design level. Each requirement at the system level, describing necessary functionality and the functional specification of module interfaces and algorithm input variables, is considered for the following: If it performs the function will it cause a hazard; if it fails to perform the function will it cause a hazard; if it performs the function in the wrong order or sequence will it cause a hazard; if the function is active too long or finishes too soon will it cause a hazard. The complete system operating environment and modes of operation are considered while performing this assessment.

Keywords: Software Safety Assessment, Software Validation, Diagnostics, Software Hazard Analysis.

Acknowledgement: I would like to thank Lyn Fernie (Business Manager at HIMA Consultancy) for her support, without which this paper would have not been possible.

1 Introduction

Software Safety Assessment is often limited to performing a Software HAZOP (Hazard and Operability Analysis) at the System Requirements level, and by the review of the Software lifecycle phase outputs to determine whether the design implementation and the Software Development process followed meet the requirements of the appropriate Safety Integrity Level (SIL) in standards such as IEC61508:2011 (International Electrotechnical Commission, 2011, Functional safety of electrical/electronic/programmable electronic safety-related systems). The SIL identified for a System is the amount of risk reduction required to bring the System risk within a tolerable level. IEC61508:2011 contains tables detailing a number of Highly Recommended and above requirements for the techniques/methods that are to be included in the Software design implementation and development lifecycle based on the identified SIL for the Software functions.

The limitations in the traditional approach to Software Safety Assessment may result in gaps in identifying mission/safety critical Software functions, which may in turn lead to gaps in the definition of the system response to erroneous or invalid states due to input variables. This can lead to loss of human life and considerable financial costs. An example is the MARS Climate Orbiter developed by Lockheed Martin on behalf of NASA, and is based on the findings of the incident investigation report (Mars Climate Orbiter Mishap Investigation Board, 1999).

The Orbiter was launched on the 11th of December 1998 to study the Martian climate. After reaching Mars the Orbiter began its orbital insertion where upon communication was lost with the craft. A later investigation found that a piece of ground equipment software (application SM_FORCES) produced readings in "Imperial" units that represented thruster performance data. A second piece of equipment software (MSOP Project Software Interface Specification) expected the readings to be in metric units. As a result the impulse produced by the thruster firings were calculated in pound-seconds rather than the intended newton-seconds. The discrepancy led to an incorrect orbit insertion altitude which resulted in the Mars Climate Orbiter encountering Mars at a lower than anticipated altitude and disintegrated due to atmospheric stresses.

In an attempt to avoid such errors, an approach for identifying additional software hazards and their resultant control/mitigation at three levels is proposed. By taking this layered approach the Safety Engineer will be able to improve the definition of the system response to erroneous or invalid system states ultimately leading to more robust architectures that are able to safely respond to these conditions. The approach will also result in the identification of the mission/safety critical functions that are to be assessed in greater detail. The three levels are:

- System Requirements
- Software Architecture
- Detailed Design (interface between functions/modules and safety/mission critical input variables for algorithms)

The approach detailed in this paper also forms part of the required safety activities in the following bulleted phases of the IEC61508:2011 lifecycle:

- Hazard and risk analysis
- Overall Safety Requirements
- E/E/PE System Safety Requirements Specification
- Realisation
- Overall Safety Validation

The main scope of this paper is the technical aspects of Software Safety Assessment with particular attention being given to the implementation of a technique using a modified Software HAZOP and Software Error Analysis at the System Requirements, and Detailed Design levels. An intermediate level Software Architecture analysis using Fault Trees is also covered. The Fault Tree analysis of the architecture is to ensure the software functions that contribute to unwanted events are identified so that the Safety Engineer can target an in depth analysis of these functions.

The structure of this paper has been divided into the following sections:

Section 2: Covers the Software Function Analysis at the System Requirements level. This approach is similar to a Software HAZOP in that a team of suitably competent personnel will use guidewords to direct the assessment of the descriptions of functionality in the System Requirements Specification. The output of this phase will lead to high level safety requirements used to control causes of hazards.

Section 3: Covers a Fault Tree Analysis at the Software Architecture level. It specifically outlines how to interpret process and state diagrams to produce Fault Tree models for the system. The outputs of this phase will lead to the identification of the Software Functions that contribute to unwanted events that are to be assessed in greater detail during the Detailed Design assessment level (Section 4). Note that the SIL targets of the Software Functions can also be determined using the Fault Tree Models, however this is not the topic of this paper.

Section 4: Covers the Software Function Analysis at the Detailed Design level. This approach is similar to a Software HAZOP and Error Analysis in that the Safety Engineer will use guidewords to direct the assessment of the safety/mission critical interfaces between software functions and input variables to algorithms. The application of this technique at the Detailed Design level is almost always omitted from traditional Software Safety Assessment. The output of this phase will lead to additional requirements defining the system behaviour when erroneous or invalid system states occur.

During a project development a number of other techniques and processes must be implemented to control risks due to systematic error, to reduce the probability of random failures occurring, and to meet the lifecycle phase requirements of the relevant standards. Due to the complexity of each topic area and the need to produce a standalone document specifically covering each in detail, these topics will not be covered in detail in this paper. It should be noted by the reader that the omitted topic areas must meet the phase requirements of the relevant Safety and Software lifecycle models for the project and should be researched in depth before attempting to manage and implement a Software Safety programme. These topic areas include but are not limited to:

- SIL Assessment
- Development and implementation of a Safety Management System
- Development and implementation of a Quality Assurance Programme and Quality Management System
- Development and implementation of Verification and Validation Strategies for each phase of the lifecycle
- Software Safety Strategy and Assessment Planning
- Competency Management and Assessment
- Requirements Traceability and Configuration Management over the complete lifecycle

It is to be noted that the high level approach described in this paper is similar, in part, to an FHA described in ARP4761 (SAE International, 1996, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment) in that descriptions of system functions (including purpose and behaviour) are assessed against hypothetical failure modes, whereby the effects and risk levels on the system are then determined. Furthermore, the approach of using default behaviour conditions, assessed against functions treated as components of a system originates from a traditional FMEA.

2 Software Function Analysis – System Requirements Specification level

The Software Function Analysis at this level specifically targets the function descriptions detailed in the System Requirements Specification and uses a guideword approach similar to a Software HAZOP. Each identified requirement is assessed against behaviour condition descriptors to lead the assessment team to consider under which behaviour condition the function will present a hazard. The behaviour conditions selected originate from the book 'Engineering a Safer World' (Leveson, Nancy G, 2011), and are as detailed in Table 1.

Table 1 - Behaviour conditions

Behaviour condition	Description/interpretation
Performing the function causes a hazard	Under what operating conditions or modes of operation of the system will performing this function cause a

	hazard?
Not performing the function causes a hazard	Under what operating conditions or modes of operation of the system will not performing the function cause a hazard?
Performing the function in the wrong sequence or at the wrong time causes a hazard	If the function is carried out in the wrong sequence or at the wrong time in the process under what operating conditions, modes of operation of the system will it cause a hazard?
The function being stopped too soon or active too long causes a hazard	If a function is carried out, but the function does not finish a complete cycle, or the function is held active after it should have finished a cycle, under what operating conditions or modes of operation of the system will it cause a hazard?

The Safety Engineer is responsible for populating the Function Analysis assessment worksheet and will distribute it amongst the assessment team prior to the workshop taking place. This is most efficiently achieved by performing a requirements export using a requirements management tool e.g. DOORS, Eigner SKM, JAMA Contour. A good approach is to group the exported requirements into the days on which they are to be reviewed. This provides a useful tool to manage progress and will ensure requirements are reviewed in manageable portions thereby reducing the risk of systematic error due to team fatigue.

The Safety Engineer is responsible for selecting the assessment team based on the competencies required for the activity. The competency requirements should be identified and agreed prior to selecting the team ideally using a competency framework from a recognised institute as a guide. Competency management is a detailed topic area and is outside of the scope of this paper. Once the team has been selected the Safety Engineer should brief the members on the Software Safety approach prior to the workshop commencing and with particular attention to the Software Function Analysis

The Safety Engineer will be in the role of the facilitator during the workshop and will guide the team through the assessment of each function in turn. The members of the assessment team will consider the mode of operation and operating environment of the system and shall identify the impact of each behaviour condition occurring. Where a hazard is identified it is recorded in the spreadsheet by the Safety Engineer. Each function is to be assessed initially to identify new hazards that contribute to existing accidents/high level unwanted events. It is critical to maintain traceability in the spreadsheet between the accidents, hazards, causes and mitigation. As a result the unique hazard identifier shall be recorded in the worksheet traceable to the high level accident (this will support the Fault Tree process in section 2).

Once this is complete the assessment team shall review each hazard entry to identify whether there are existing requirements to provide sufficient control/mitigation. If existing requirements are in place the Safety Engineer shall record the unique reference indicator in the spreadsheet to maintain traceability. Where a new requirement is needed to mitigate/control the hazard a new requirement entry is to be entered into the spreadsheet. It is to be noted that the review team can begin defining the new requirement during the Software Function Analysis, although experience has indicated that it is best tackled outside of the workshop by the developers with support from the Safety Engineer. At this stage all that is required is a record that a new requirement is needed and some commentary regarding the proposed functionality.

On completion of the worksheet the Safety Engineer shall produce a Software Function Analysis Report that is to be reviewed by the assessment team, and reviewed and approved by a Safety professional with a strong background in Systems Engineering. Once approval has been given the Safety Engineer should make a copy of the original worksheet and modify it to enable an import back into the Requirement Management Tool. This import is to ensure that the System Requirements are linked to the identified hazards (in principle the hazards will become requirements within the Requirement Management Tool), and subsequent mitigation/control requirements are linked to the hazards. All mitigation/control requirements should be tagged as safety requirements. Where new requirements are needed the Safety Engineer should tag the system requirement link as being suspect and generate a Requirements Traceability Report. The Requirements Traceability Report and the Software Function Analysis Report will be used as the key drivers for ensuring all new requirements are generated to describe the system's response to ensure a safe state is achieved when a behaviour condition occurs leading to a hazard.

An example of a Software Function Analysis at the System Requirements and Detailed Design Level has been provided in Table 2. The second row of Table 2 is the assessment of Function A in the System Requirements Specification. The function is responsible for calculating the brake pressure for a hypothetical train service brake system.

In this example each Behaviour Condition has been assessed against the function by taking into account the operating modes of the system and the operating environment. It can be seen in the table that behaviour condition 4b) assessed for Function a "calculate brake pressure" results in a hazard "Exceeding the stopping distance". The rationale behind this is because in the hypothetical system if the brake pressure calculation function is held active without a new function cycle commencing, the sensor reading data would not be updated which would potentially lead to an erroneous brake pressure being set. Further investigation would have identified that the current requirements do not contain sufficient mitigation for this hazard, therefore the assessment team would record that a new requirement is needed and include commentary regarding its functionality see REQ1.7 and REQ1.8 in Table 2. The new requirements can eventually be used to define additional states or data flow entries in state and data flow diagrams produced by the development team and presented in the Detailed Design Description.

It is important to note that the Software Function Analysis must be controlled under the configuration management system. The applicable version of the

System Requirements Specification used in the assessment should be recorded and a mechanism/process implemented to manage updates due to failures or issues found over the complete lifecycle of the system. During the development phase any updates to the System Requirements should trigger a review of the applicable section in the Software Function Analysis.

After completing the Software Function Analysis at the System Requirement Specification level, the Safety Engineer can use the outputs along with the System Requirement Specification, Architecture documentation and initial Detailed Design Descriptions (in particular State and Data flow diagrams) to begin modelling the Software Architecture with Fault Trees. The aim with this approach is to identify the safety/mission critical software function contributions to unwanted events and ultimately the areas of the design that should be assessed in greater detail during the Detailed Design assessment level e.g. system response to erroneous or invalid algorithm input variables and function/module interfaces. The use of Fault Trees to define the SILs for software functions is outside of the scope of this paper, however they should typically be identified at this stage of the assessment process.

3 Software Architecture Analysis – Fault Tree Modelling

Typically in a development once the System Requirements have been reviewed and agreed, the development team will begin to identify additional derived requirements and produce the detailed design description. The detailed design description will typically include the state and data flow diagrams defining the system logic and behaviour response to events and conditions with full traceability to the Software Architecture.

The Safety Engineer can use the information in the Detailed Design Description, the System Requirements Specification, the Architecture block diagram and descriptions, and the outputs of the Software Function Analysis (Requirements Specification level) to produce a Fault Tree.

The Fault Tree models can be combined into an overall System Fault Tree (including hardware failures) to perform a complete cut-set analysis). The strengths of this approach are that the architecture is modelled in a way that is easily interpreted by Safety Case reviewers and stakeholders, and it will typically identify the combinational failures of Software functions that lead to an unwanted event occurring to be investigated further during the Software Function Analysis at the Detailed Design level. The Safety Engineer using this approach should be able to identify and recommend additional functions such as self-test and diagnostics, and be able to determine the software integrity levels for safety/mission critical functions identified as contributors.

The diagram shown in Figure 1 is an example of the mapping of a simple Process Data Flow Diagram for a brake pressure calculation process for a train service brake system to a Fault Tree model. The Safety Engineer should follow a recognised Fault Tree guide/standard when producing the model to ensure descriptors, level structures and calculation models are correct. In this example the Brake Pressure Calculation data flow diagram has been modelled as a Fault Tree. The Brake Pressure Calculation algorithm has the following main inputs:

- Train Configuration Data (including train type and brake valve configuration)
- Tachometer reading from System A (independent system)
- Tachometer reading from System B (independent system)

Industry experience has shown that typical software failure conditions reveal themselves as an error condition on the process output flows. Often the output errors are a direct result of algorithm errors in the process itself or by errors in the data flow inputs into the process. It can be seen from Figure 1 that the main contributors to the unwanted top level event are that the "Train Configuration" and the "Brake Pressure Algorithm" are incorrect, as a result they will require further investigation using the Software Function Analysis at the Detailed Design level. Note that the Fault Tree also highlights assumptions that the Tachometer inputs are provided by the independent systems will need to be justified.

Once the critical contributors have been determined they should be investigated in greater depth at the Detailed Design Level using the Software Function Analysis approach initially described in section 2 and further expanded in section 4. The Safety Engineer should identify the specific interfaces between each function and the input variables to safety/mission critical algorithms and use this information to add new entries into the original Software Function Analysis spreadsheet (to be reviewed at a later workshop). Note that because the Fault Tree structure has provided an initial framework for identifying the Software functions to be investigated further at the Detailed Design level, this should reduce the level of complexity in managing the overall assessment process as the Safety Engineer can take a methodical approach of analysing each of the Fault Tree level safety/mission critical functions in turn.

Where State Diagrams are to be mapped to a Fault Tree, a similar approach to the Process Data Flow Diagram can be used. For reasons of simplicity the example provided in Figure 2 has not been entered into the Software Function Analysis at the Detailed Design level in Table 2. The intention of the Fault Tree is to identify the conditions under which the top action becomes true. It can been seen in Figure 2 that the main contributors to the unintended removal of the Emergency Brake are condition 3 occurring, or Conditions 1 and Condition 2 occurring when not intended. These conditions will need to be investigated further during the Software Function Analysis at the Detailed Design level. A closer investigation of Condition 3 might directly lead the Safety Engineer to identify the hardware failures in operation and human errors during maintenance that can lead to condition being true e.g. random failure of the communication and power isolation relays, maintenance staff leaving control bypass mechanisms in place, or an incorrect software version being uploaded. On closer investigation of Condition 1 and 2, the Safety Engineer may identify specific command signals that if they behave erroneously will directly lead to the condition being met e.g. an erroneous command to activate the communication relay, and an erroneous command to remove the system operational inhibit.

Once the Safety Engineer has completed the Fault Tree modelling of the Software Architecture, and has identified all safety/mission critical functions, interfaces and algorithm input variables, new entries are to be added into the original Software Function Analysis worksheet. A new Software Function Analysis can then be performed at the Detailed Design level to review the new entries and to identify additional hazards and the need for new mitigation/control requirements.

4 Software Function Analysis – Detailed Design Level

The outputs of the Fault Tree Analysis at the Software Architecture level have identified critical areas of the software design that will require further investigation. Expanding on the assessment in section 2, the Safety Engineer can identify the safety/mission critical interfaces between functions, and algorithm input variables using the information contained in the System Requirement Specification and the Detailed Design Description. This information is then added by the Safety Engineer to the Software Function Analysis spreadsheet as a new function entry. Note that the process for performing the Software Function Analysis is identical to that described in the Requirements Specification level in section 2 above.

The aim of this level of assessment is to flush out the final requirements to define the system behaviour to erroneous or indeterminate system states due to errors and failures at the function interfaces and input variables to safety/mission critical algorithms. As a result, a more robust design will be created that can safely manage erroneous or invalid system states.

In the example in Table 2 the Brake Pressure Algorithm (identified as safety/mission critical in the Fault Tree in Figure 1) is investigated further. A review of the design specifications and documentation identifies there are three input variables to the Brake Pressure Algorithm, these are the following:

- Train Configuration data – Variable X
- Tachometer reading A – Variable Y
- Tachometer reading B – Variable Z

The Safety Engineer will use this information to populate the Software Function Analysis spreadsheet with the Brake Algorithm as a function with three input variables. The Safety Engineer will organise a workshop (as described in the Requirements Specification level analysis above) to assess each new entry (in this example the input variables to the algorithm) against the default behaviour conditions to determine whether it will result in a hazard. Where hazards are identified the Safety Engineer will determine whether existing mitigation exists or whether additional protection is required to defend against invalid or erroneous states due to input errors.

It can be seen from the results of the Software Function Analysis at the Detailed Design level in Table 2 the incorrect order of the variables used in the Brake Pressure Algorithm may directly lead to the hazard "Exceeding the stopping distance". The need for a new requirement has been identified see REQ1.9, including initial commentary describing the functionality in the attempt to govern the system response to a previously invalid and erroneous system state. This layered approach to Software Safety assessment typically leads to an incremental improvement of the robustness of the system response behaviour and it is believed that it will ultimately lead to safer system architectures.

On completion of the workshop the Safety Engineer should follow a similar process to that described in section 2 at the System Requirements level with regard to requirement management, traceability, and producing a final report that is to be reviewed and approved, and will be used as a driver for the outstanding requirements to be defined and implemented.

5 Conclusion

The complexity of software intensive systems requires a more in depth approach to functional safety assessment. To date, numerous incidents have occurred across industries due to poorly defined system behaviour, poorly understood emergent behaviour, and gaps in the identification of safety/mission critical functions using a traditional two layered approach of performing a Software HAZOP and SIL verification. The three layered approach proposed in this paper has the potential to identify additional system behaviour requirements to manage and improve the exception handling performance characteristic and provides a technique to manage the complexity of software safety assessment for complex software intensive systems. The approach provides a methodical and structured technique for identifying hazards at the System Requirements level, identifying the safety/mission critical software functions at the architecture level, and identifies the hazard associated with safety/mission critical functions at the detailed design level. Implementing the approach as described in this paper will ensure the identified hazards are managed over the lifecycle and unwanted design features are captured systematically and removed before the system enters service. Project experience and in service reliability and safety records of systems using this approach has led to the conclusion that the three layered approach is superior compared to assessing the functional safety for software using only a Software HAZOP and SIL Verification.

Current limitations with the approach concern the resources required to perform the analysis at the detailed design level. To reduce the resource demand systems engineering techniques should be implemented such as the use of a Requirements Management Tool to automatically generate the requirements to be assessed, and to manage the Hazards as objects with full traceability from System Requirements, to accidents, to hazards, to causes, to design implementation, through to design specifications and test reports.

In summary, the approach described in this paper, if implemented alongside a Verification and Validation programme across all phases of the system lifecycle, will support an accurate and in depth assessment to determine the true safety performance characteristic of a System's software. It will also support the argument that a Functional Safety Management and Quality Management System of sufficient rigour has been implemented, and the system meets the requirements of the SIL This in turn will increase end user confidence in the delivered system and its safety (including exception handling) performance characteristic.

6 References

International Electrotechnical Commission. IEC61508:2011: Functional safety of electrical/electronic/programmable electronic safety-related systems.

CENELEC. EN50126:2012: Railways Applications – The Specification And Demonstration Of Reliability, Availability, Maintainability And Safety (RAMS)

SAE International. ARP4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.

Leveson, Nancy G. (2011): Engineering a Safer World: Systems Thinking Applied to Safety. Cambridge, Massachusetts, London, England, MIT Press

Mars Climate Orbiter Mishap Investigation Board (1999): Phase 1 report

7 Tables and Figures

Table 2 - Software Function Analysis - System Requirements and Detailed Design levels

Function	1) Performing function causes a hazard	2) Not performing function causes a hazard	3) Performing function in the wrong sequence or at the wrong time causes a hazard	4) a) Function stopped too soon, 4) b) or active too long causes a hazard	Existing Requirement	New Requirement
(Requirement Specification) Function A - Calculate Brake Pressure (Traceable to REQ 1 in the System Requirement Specification)	If input sensors have failed or are outputting erroneous data the service brake pressure calculated will be incorrect – Leads to "Exceeding the stopping distance" (Add an Accident Ref, a Hazard Ref, and add a Cause Ref, and enter them into the Requirements Management Tool)	A brake demand is received but the brake pressure is not set the service brake will not activate – Leads to "Exceeding the stopping distance (Add an Accident Ref, a Hazard Ref, and add a Cause Ref and enter them into the Requirements Management Tool)	If brake pressure is calculated before an accurate and up to date reading is taken from the input sensors the service brake pressure calculated may be incorrect - Leads to "Exceeding the stopping distance" (Add an Accident Ref, a Hazard Ref, and add a Cause Ref and enter them into the Requirements Management Tool)	4) a) The brake pressure will not be calculated after a service brake demand is received, the service brake will not be set – Leads to "Exceeding the stopping distance" (Add an Accident Ref, a Hazard Ref, and add a Cause Ref and enter them into the Requirements Management Tool) 4) b) The read cycle from the input sensors would not be updated, the processor cycle will potentially be held up leading to erroneous service brake pressures being calculated and the loss of time critical functions – Leads to "Exceeding the stopping distance" (Add an Accident Ref, a Hazard Ref, and add a Cause Ref and enter them into the Requirements Management Tool)	1) REQ1.1 – Sensors undergo self-tests to detect failures. REQ1.2 – Sensors are redundant and are voted for accuracy reasons. 2) REQ1.3 – If a brake demand is given but the service brake is not set the Emergency Brake will be activated. 3) No existing requirement, a new one is needed 4) a) REQ1.3 4) b) No existing requirement, a new one is needed	3) REQ1.5 – A new reading from the sensors must be received before the service brake pressure is calculated. 3) REQ1.6 – If a sensor reading is not received after a service brake demand is given, a default maximum service brake pressure will be set using a memory protected data source 4) b) REQ1.7 – After the successful engagement of the brake the brake pressure calculation is monitored to ensure the calculated service brake pressure is reducing during the braking cycle. 4) b) REQ1.8 – If the calculated service brake pressure fails to reduce during a braking cycle a fault is flagged and the maximum service brake pressure is set using default data values from a memory protected data source.
(Detailed Design) Function A – Brake Algorithm uses input variables X, Y and Z to calculate brake pressure.	Calculating brake pressure using variables X, Y and Z do not cause a hazard	A brake demand is received but the brake pressure variables are not set, as a result the service brake will not be activated – Leads to "Exceeding the stopping distance" (Add an Accident Ref, a Hazard Ref, and add a Cause Ref and enter them into the Requirements Management Tool)	If variables X, Y and Z are sent in the wrong order an incorrect brake pressure will be calculated - Leads to "Exceeding the stopping distance" (Add Hazard Ref)	4) a) The brake pressure variable will not be set, as a result the service brake will not be activated – Leads to "Exceeding the stopping distance" (Add Hazard Ref) 4) b) The read cycle from the input sensors would not be updated, the processor cycle will potentially be held up leading to erroneous service brake pressures being calculated and the loss of time critical functions – Leads to "Exceeding the stopping distance" (Add Hazard Ref)	2) REQ1.3 – If a brake demand is given but the service brake is not set the Emergency Brake will be activated. 3) No existing requirement, a new one is needed 4) a) REQ1.3 4) b) REQ1.8	3) REQ1.9 – Variables X, Y and Z are monitored to determine whether in the correct order. If variables are in the incorrect order a fault is flagged and all three variables are rejected. The maximum brake pressure is then set using three known variables from a memory protected data source.

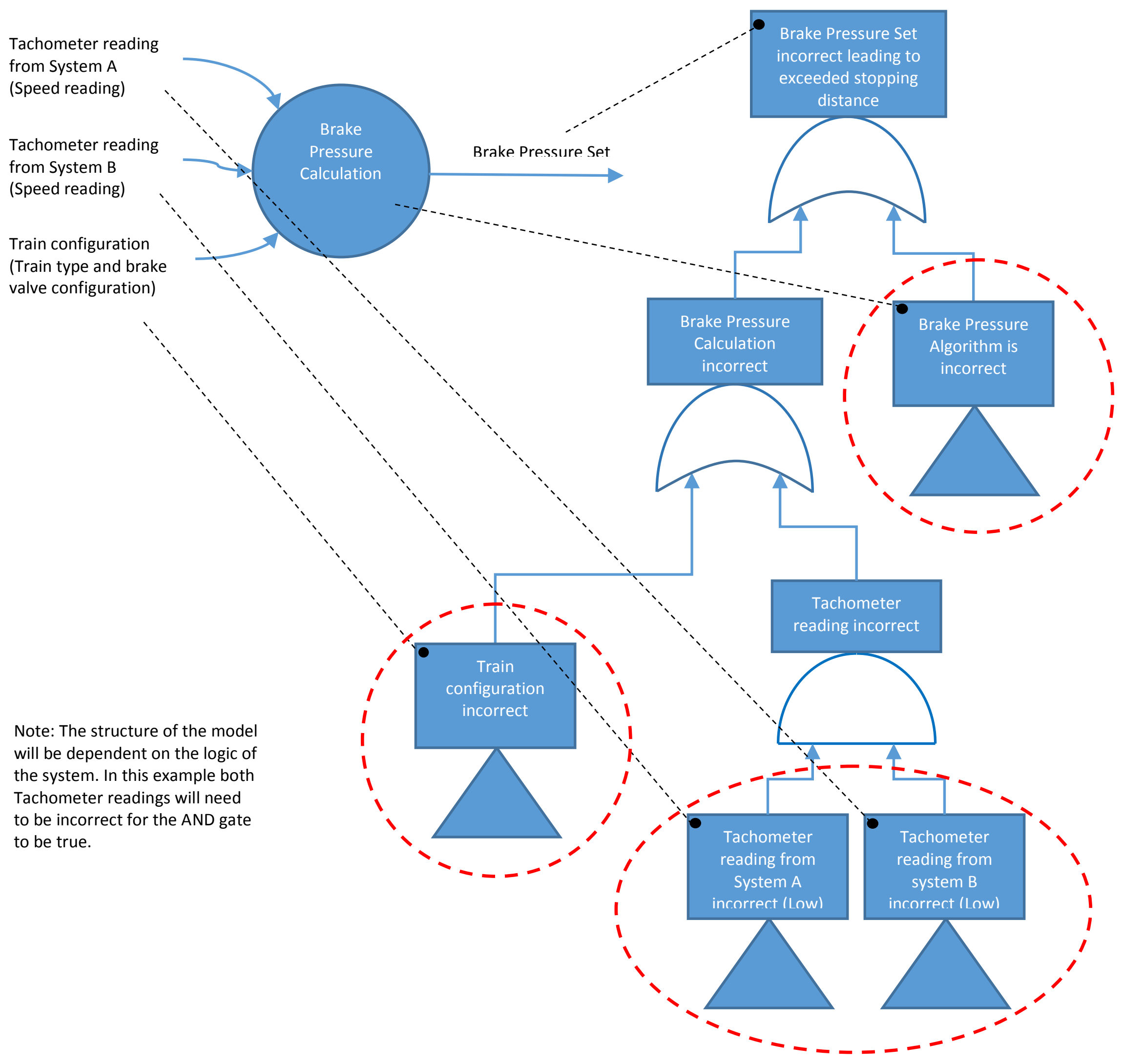

Tachometer reading from System A (Speed reading)
Tachometer reading from System B (Speed reading)
Train configuration (Train type and brake valve configuration)
Brake Pressure Calculation
Brake Pressure Set
Brake Pressure Set incorrect leading to exceeded stopping distance
Brake Pressure Calculation incorrect
Brake Pressure Algorithm is incorrect
Tachometer reading incorrect
Train configuration incorrect
Note: The structure of the model will be dependent on the logic of the system. In this example both Tachometer readings will need to be incorrect for the AND gate to be true.
Tachometer reading from System A incorrect (Low)
Tachometer reading from system B incorrect (Low)

Figure 1 – Simple Process Fault Tree

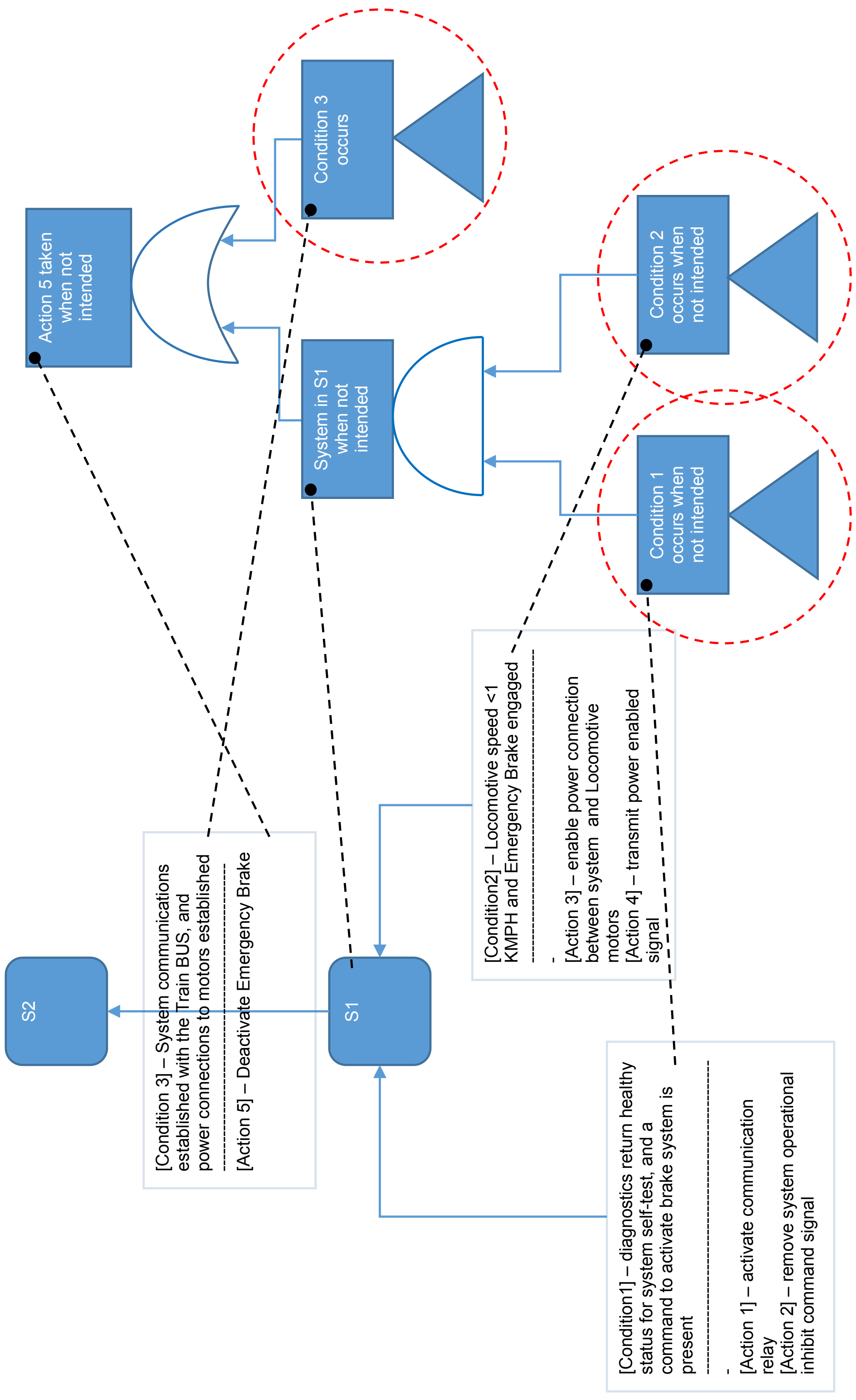

Action 5 taken when not intended
Condition 3 occurs
System in S1 when not intended
Condition 2 occurs when not intended
Condition 1 occurs when not intended
[Condition 3] – System communications established with the Train BUS, and power connections to motors established
[Action 5] – Deactivate Emergency Brake
S2
S1
[Condition2] – Locomotive speed <1 KMPH and Emergency Brake engaged
-
[Action 3] – enable power connection between system and Locomotive motors
[Action 4] – transmit power enabled signal
[Condition1] – diagnostics return healthy status for system self-test, and a command to activate brake system is present
-
[Action 1] – activate communication relay
[Action 2] – remove system operational inhibit command signal

Figure 2 – State Diagram Fault Tree

Independent Safety Assessment – White Paper

Kevin J Anderson and Tariq Mahmood
Evans & Peck Advisory Consultants
Level 15, 607 Bourke Street
Melbourne VIC 3000 Australia

kanderson@evanspeck.com, tmahmood@evanspeck.com

Abstract

This white paper describes the rationale for Independent Safety Assessment (ISA). Pros and cons are assessed and conventional wisdom is challenged – hence the phrase White Paper to denote understanding the issues, solving any problems and making a decision – ISA or no.

The content of the paper proceeds from a consideration of relevant standards AS 61508 - Functional safety of electrical, electronic and programmable electronic safety-related systems' - Edition 2 – 2010 and EN 50126 – Railway applications – Specification and demonstration of reliability, availability, maintainability and safety (RAMS) - 1999.

The software life cycle is reviewed in relation to the common law tests of negligence: *causation, foreseeability, preventability* and *reasonableness*.

A software-related case study is then presented relating to computer-based train orders.

Keywords: reliability, availability, maintainability, safety, causation, foreseeability, preventability reasonableness.

1 Introduction

The science of reliability stemmed from defence industry review of WWII technology. Of course there were successes but also many failures. Aerospace picked up on the techniques with space missions. Since 1991 the UK Ministry of Defence has required an Independent Safety Auditor to provide the independent, objective opinion of safety that was lacking in defence projects at that time. (Froome, 2005)

A raft of dependability standards were introduced in the 1990s and are being continually refined to this day. Refer AS/IEC 61508 – 2010. AS 61508-2010 is a normative reference prescribed by the railway application EN 50126-1999. That is, AS 61508 overrules and is to be preferred to EN 51026 – Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS), and the sisters EN 50128 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems and EN 50129 - Railway applications - Communication, signalling and processing systems - Safety-related electronic systems for signalling. In all of these standards, safety is defined as 'freedom from unacceptable risk of harm' and presages a risk-based approach. The definition of 'audit' includes 'systematic and independent examination…'

In the United Kingdom the principle of reducing risk 'as low as reasonably practicable' (ALARP) has been applied. In France 'Globalement au moins aussi bon' (GAMAB) – at least as good as any equivalent system and Germany 'minimum endogenous mortality' (MEM).

The International Engineering Safety Management (IESM) Good practice handbook (2013) replaces the now withdrawn rail industry Yellow Book (2007). At 3.7.2 it states that 'engineering safety management activities must be reviewed by competent people who are not involved with the activities concerned.

An independent assessment provides that second opinion. The degree of independence and the rigour of the assessment depend on complexity, the extent of the risk and the novelty of the system.

AS 61508 provides additional guidance to EN 50126 using the term 'Functional Safety Assessor' (FSA) as relevant to electrical, electronic /programmable electronic (E/E/PE) safety related systems. FSA is taken here to be a subset of the ISA role. The discussion below is given in the context of the rail industry.

The role of the FSA is to investigate and arrive at a judgement on the adequacy of the functional safety of E/E/PE safety-related systems, sub systems elements and items. Requirements include:

- access to all relevant persons and equipment;
- application to all lifecycle phases;
- all relevant claims of compliance made by suppliers and other parties;
- assessment of evidence from functional safety audits;
- recommendations from previous assessments;
- planning to facilitate effective assessment;
- approval of the FSA Plan by those responsible for functional safety;
- documentation;
- facilitation of re-use of assessments;
- competency;
- minimum level of independence .

Independence is a function of consequence, safety integrity level (SIL) and systematic capability and experience with a similar design complexity and novelty.

The FSA can be an independent person, independent department or independent organisation.

In Australia, the Rail Safety National Law establishes the Office of the National Rail Safety Regulator (ONRSR) and deals with rail safety duties, accreditation, registration, safety management, information and audit.

Later parts deal with compliance, enforcement, exemptions, review and liability. The State Acts, for example Rail Safety National Law (NSW) require risk reduction ..'so far as is reasonably practicable' (SFAIRP) – an arguably more stringent test of 'as low as reasonably practicable' (ALARP).

This paper reviews the role of the Independent Safety Assessor (ISA) in both the legislative and standards context. The RAMS life cycle is also reviewed in relation to the legal duty and common law tests of negligence: *causation, foreseeability, preventability* and *reasonableness*.

Key questions are:

- What is the role of an ISA?
- What does an ISA do?
- Why do I need an ISA?
- What are the risks of engaging /not engaging an ISA?

The answers to these questions depend on an understanding of a particular project life cycle and its approval and contractual arrangements.

In the absence of rail safety legislation and an approval authority, the principal (customer /operator) has to establish requirements and carry out approval, validation and acceptance activities. A contractor, supported by sub-contractors and suppliers will implement solutions and also contribute to validation.

2 Life Cycle Phases

Independence is defined at 3.3 of EN50126 in terms of audit (of the Safety Plan).

EN50126 (clause 6.2.3.4 (f) safety audit) requires that independent safety assessment of the railway system, its sub-systems, equipment and its safety cases is undertaken to provide assurance that the necessary level of safety has been achieved.

A Safety Management System (SMS), where such is established by the Railway Authority, would also be expected to provide standards and guidelines for ISA.

EN50126 sets out 14 life cycle phases and Annex E assigns responsibilities between Principal and Contractor.

The role of the ISA is considered here in relation to four groupings:

- Concept and scope
- Risk analysis and requirements
- Realisation and acceptance
- Operations and maintenance

2.1 Concept and Scope

The Principal is expected to develop the concept, system definition and application conditions. Safety was once seen as largely the Contractor's responsibility, but now rests with the 'Duty Holder', for example, under Rail Safety National Law (NSW).

Froome (2005) notes that while an ISA's opinion must be sought on the quality of the safety case, the ISA has no executive authority or power of veto. ISA recommendations can be overruled.

The *causation* principle requires demonstration of a level of understanding sufficient to support the life cycle activity.

Pursuant to a documented SMS or no, a Safety Plan (SP) must be prepared in any case and maintained throughout the life cycle.

At this phase, the ISA would be engaged in initial review of safety requirements as reflected in the SP.

Under AS 61508-1 clause 8.2.8, a functional safety assessment must itself be planned and the plan approved by those responsible for safety. Considerations include scope, resources, level of independence (based on consequence and/or safety integrity level), personnel, competence and expected outputs.

2.2 Risk analysis and requirements

Risk assessment is a core safety activity in most SP elements.

The risk analysis, system requirements and apportionment are shared between the Principal and the Contractor. It is expected that these will simultaneously consider issues (foreseeability) and options (preventability).

The ISA will review safety assurance documentation including safety cases, the project hazard log and risk register.

2.3 Realisation and acceptance

Reasonableness is the hard part, reaching a consensus between the significance of a risk and the effort /cost to control it.

The Contractor will primarily carry out the design and implementation, manufacture and installation.

It is expected that the safety cases will be further developed and the SP updated accordingly. It is expected that the safety hazard log and risk register will evolve into agreed safety requirements to be matched against acceptance criteria.

A program of safety assurance audits is required to ensure that Contractor(s) comply with the SP and that work is carried out safely.

The safety cases will be reviewed and amended until a status of 'not rejected' is arrived at.

An Independent Certifier (IC) may be engaged to observe, monitor, audit and test all aspects of the quality and durability of the work and to verify that the Contractor is complying with all requirements and that works have been satisfactorily completed.

The IC will interact with the ISA.

System validation is a joint activity and system acceptance is up to the principal.

2.3.1 Verification and Validation tasks

Verification and Validation (V&V) are respectively confirmation by examination and provisions of objective evidence that:

- the specified requirements have been fulfilled (verified); and
- the particular requirements for a specific intended use have been fulfilled (validated).

The ISA will ensure that:

- the EN 50126 process has been carried out effectively;
- all required safety activities and arguments are adequate as to scope and quality;

- all safety requirements are satisfied; and
- safety assurance deliverables are acceptable.

2.4 Operations and maintenance

The Principal will operate and maintain the system including performance monitoring, modification /retrofit and de-commissioning /disposal.

3 Case study

A case study is presented below for the NSW Country Rail network, based on the authors' experience and opinion, one of whom has been the appointed FSA /ISA for this system since 1994.

A software based Train Management and Control System (TMACS) has been deployed with the GPS Watchdog component certified as ISA by one of the authors to SIL 2. TMACS has been in service for 15 years in western NSW and has recently been rolled out across the entire north and south NSW country networks. Further software changes and re-certification are in progress to accommodate In-cab-equipment (ICE) trains.

The following sections present, firstly, a description of the compliance of the Watchdog to limit the dangerous failure rate and to achieve necessary independence and, secondly, compliance with software development measures and techniques.

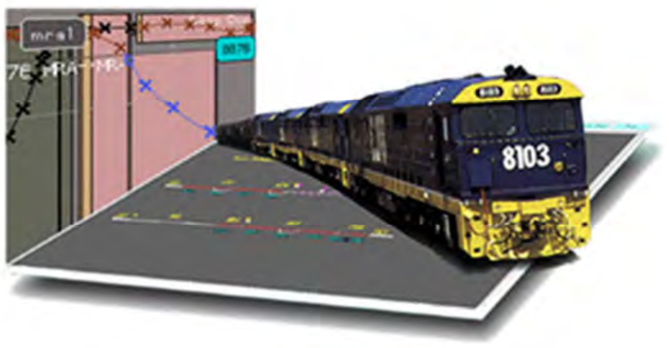
3.1 TMACS GPS Watchdog

The Train Management and Control System (TMACS) is a form of Train Order Working involving safeworking encodings and GPS train location data. Services commenced in 1999 in the Orange –Parkes –Dubbo – Broken Hill areas.

The Train Order case study began in 1994 with the introduction in Victoria of the now superceded 'Alternative Safeworking System' (ASW). A software failure led to two trains in the one section - the 'loss of control' situation that ASW was designed to prevent. Only quick thinking in the field saved a collision.

The fault was attributed by the investigating ISA to unwarranted reliance on n-version programming and lack of visibility of the software design process to the customer. The software recovery process recommended by the ISA is now discussed.

Following the software failure described above, the ISA decided not to declare the NSW derivative of ASW the 'Train Management and Control System' (TMCS) as safety-related. Rather, under Part 1 of AS 61508 (the Standard) clause 7.5.2.6, TMCS was designated by the ISA as Equipment-under-control (EUC) and a separate safety-related GPS Watchdog (GPSW) was instituted. Given the role of the ISA in recommending this approach, a second independent organisation was engaged to review the ISA final documentation prior to commencement of operations. The combination of TMCS and GPSW is now known as TMACS and/or the Train Order System (TOS).



In support of this declaration, the standard requires evidence of claims as to the EUC dangerous failure rate through experience, reliability analysis and /or use of a generic database and places limits on such a claim. This information was put together by the ISA

A further requirement is that the two systems be independent. Independence has a number of dimensions:

- consideration of common mode failures;
- evidence that the likelihood of simultaneous failures between systems is sufficiently low in relation to the required safety integrity;
- functionally diverse using totally different approaches to achieve the same results
- based on diverse technologies, using different types of equipment to achieve the same results;
- not sharing common parts, services or support systems (for example: power supplies) whose failure could result in a dangerous mode of failure of all systems;
- not sharing common operational, maintenance or test procedures; and
- consideration of other risk reduction measures such as other technology and external risk reduction facilities /human factors.

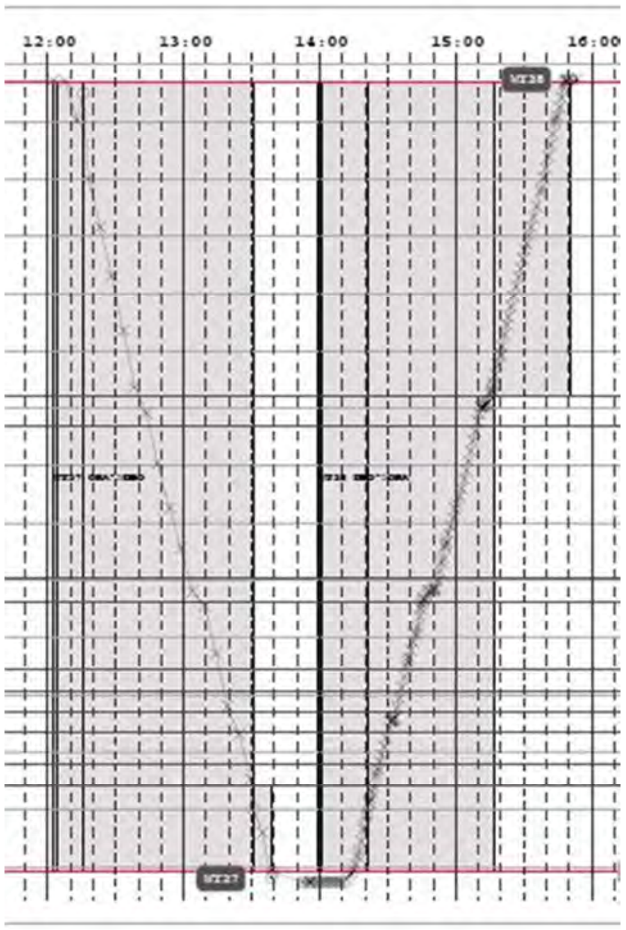
The ISA was responsible for specifying diversity for the GPS Watchdog and assisting the Railway Authority in evaluating and managing software development tenders. Compliance with each of these requirements was assessed by the ISA pursuant to part 1 clause 8. The ISA is required to investigate and arrive at a judgement on the adequacy of safety-related systems /compliant items.

A brief account of this compliance follows:

- <u>Common mode failures</u> were expressed by the ISA in cause-consequence modelling. The modelling was used to demonstrate a rare <u>likelihood of simultaneous failures</u>;
- <u>Functional diversity</u> was expressed in two ways –
    - o the representation of Safeworking Rules as data tables specific to the particular configurations of passing and shunting loops, junctions and other locations and, secondly,
    - o the presentation to the Train Controller of situational awareness, train locations and alarms on a Map in addition to the Time-Distance Graphs traditionally employed;
- Based on diverse technologies, using different types of equipment to achieve the same results:
    - o The processing of GPS position data embedded in Train Radio messages, and

  - The use of different languages and coding standards (C in the case of TMCS and Ada in the case of GPSW)
- Not sharing common parts, services or support systems (for example: power supplies) whose failure could result in a dangerous mode of failure of all systems:
- The provision of backup power systems to the Train Control Centre and development of manual and radio failure fallback procedures in the unlikely event of widespread communication blackouts, notwithstanding that satellite radio options are also in place
- Not share <u>common</u> operational, maintenance or test <u>procedures</u>. Implemented in documentation
- Consideration of <u>other risk reduction measures</u> such as other technology and external risk reduction facilities /human factors. The roles of the Train Controller, the Train Driver and Track Crew were developed through operating and emergency procedures

TMACS was rolled out in 2012 by Country Rail Network (CRN) to replace all staff working in country NSW. System enhancements in progress include Junction Location encoding and modifications to the GPS Watchdog to allow electronic authorities through the National Train Communication System (NTCS) ICE units (in-cabin equipment). The following graph shows the frequency of current position data on the left and the greatly enhanced frequency of an ICE train on the right:



3.2 Software ISA of GPS Watchdog software

This section considers the application of Part 3 of AS 61508.3 - 2011 – Software. Techniques and measures pursuant to AS 61508-7 were agreed between developer and FSA/ISA.

The results are summarised in the Annex A tables below:

A.1	Requirements at Table B.7 for semi-formal - relied on Architecture Diagram, so - low except for Rules Tables – high. Formal methods were not employed.
A.2	Design and development treated GPSW as Safety Bag. #7 C3.4 using Security Codes and Superintendent Codes to stabilise and restore the EUC modes of operation including manual, radio failure.
A.3	ADA language, coding standard and support tools, Formida GUI.
A.4	Very high modular approach, coding standards (Table B1) and structured programming.
A.5	Very high level of process simulation testing and use of boundary values (Tables B2 and B3), equivalence and functional /black box testing. Testing of all features of each 'object' as per its Detailed Requirements Specification (DRS) was by independent people within the team, FSA independent organisation – review docs and witness testing.
A.6	High level of performance and stress testing but specific techniques at Table B6 not used.
A.7	System integration test, internal validation of architecture diagram and test harness – low level data flow diagrams (Table B5), - high level of external validation of Rules Tables.
A.8	Configuration Management plan covers administration and technical controls, demonstration of systemic capability, modification requests, change-control and audit. #7 C5.23 – Impact analysis is used to assess reverification requirements – only the changed module /all affected modules /or the complete system.
A.9	Formal verification, probabilistic testing and software complexity metrics not required. Dynamic analysis and testing at Table B2 based on test logs for functions and rules. Static analysis at Table B8 limited to FSA checklists.
A.10	FSA employed failure analysis Table B4 (fault trees and event trees), checklists, Rules truth tables and reliability block diagrams.

4 Independent Safety Assessor

This section answers the key questions arising from the discussion on lie cycle phases and the lessons learned from the case study. Comments are made on improving the role of the ISA.

4.1 What is the role of an ISA

The role of the ISA is to provide assurance that rail safety is specified and implemented SFAIRP for rail systems.

The ISA shall provide an opinion (based on expert judgement of good practice) on the adequacy of safety management activities undertaken by Contractor(s).

The judgement shall consider the completeness and robustness of the system safety arguments and safety cases.

The lesson from the case study is that a pro-active and detailed effort to establish compliance with AS 61508 is required.

4.2 What does an ISA do?

ISA tasks include:

- Appreciation of the context and scope;
- Planning a cost-effective assessment approach;
- Gathering relevant evidence;
- Forming a judgement as to whether the Contractors activities are adequate and that the project safety requirements have been satisfied; and
- Managing any outcomes.

These tasks follow from compliance by the ISA to Part 1 Clause 8 of AS 61508. Under the AS 61508 lifecycle, there are sixteen phases, grouped here by example for convenience:

- Concept and scope, hazard and risk analysis, safety requirements and allocation;
- Planning, realisation, commissioning and validation;
- Operation, maintenance, modification and decommissioning.

The FSA /ISA may be carried out after each phase or group of phases and the agreed lifecycle may be different to that outlined above.

The ISA is tasked with ensuring that all hazards identified during the various phases have been independently checked and effectively eliminated or minimised SFAIRP.

The ISA audit includes review of documentation and conduct of other tasks as appropriate. For example, a RAMS analysis may include decomposition as a Reliability Block Diagram (RBD) and detailed Failure Mode Effects and Criticality Analysis (FMECA). Translation to a high level Bow-tie diagram can provide useful insights and communication to Stakeholders.

The ISA goes beyond simple reviews of contractor produced artefacts and the overall objectives are to investigate and arrive at a judgement as to the adequacy of functional safety and compliance with the standard.

4.3 Why do I need an ISA?

As above, key safety arguments include ALARP /SFAIRP, GAMAB and MEB.

Avoidance of negligence charges requires evidence that a system is safe, secure and reliable. This evidence takes the form of Safety Plans and Safety Cases supported by Safety Assurance Statements (SAS) and Safety Assurance Reports (SAR) at different phases of the project.

Appointment of an independent (third party) assessor to provide these assurances is widely regarded as international good practice. It is also demanded by the relevant standards, as discussed above..

The pros and cons of using or not using an ISA are discussed below.

4.4 What are the risks of engaging /not engaging an ISA?

Railway systems are not that special as the standards are well established and competency is readily attested.

Parties undertaking design, assurance and approvals will hold their own accreditation, authorisation and competency to discharge their functions.

The Principal and Rail Regulator, where relevant, will develop and implement SMS to effectively manage risks and safety culture.

4.4.1 Advantages of ISA

The use of an ISA is basically to increase the confidence in the conclusions of the safety assessment. The need arose in the case study from an initial risk analysis to establish that Train Orders are ‘not less safe’ than the token systems that were replaced.

The level of detail enacted arose to facilitate recovery from the initial software failure described in the case study, requiring implementation of a diverse and independent watchdog system.

4.4.2 Disadvantages of ISA

‘Extra’ cost, of course. However, this may be mitigated by accepting work conducted by other parties so as to avoid duplication of effort. In particular, IV material can be reused.

Mechanisms can also be developed to proactively prioritise resources to deal with high and very high risks and focus the level of rigour, pragmatism and independence to the degree of safety criticality.

Basic civil and structural engineering is relatively narrow in scope compared to complex safety-critical functionality, such as signalling and train control.

If an ISA is not employed, how else might confidence as to safety be achieved? Patent improvements can be defended as ‘not less safe’. AS 61508 now allows multiple routes for providing evidence of compliance. Products may already be type approved in other countries, requiring only configuration management.

4.4.3 Pros and cons of ISA

Balance is the hard part in deciding to use an ISA and at what level of engagement. Compliance with standards is but a starting point. “Best practice’ is an ephemeral target, so ‘good practice’ is defined comparatively.

5 ISA requirements

This section covers the qualifications and competence of an ISA and the scope of services to be provided, based on the authors’ experience and opinion.

5.1 Qualifications and competence of an ISA

The ISA team must be qualified, competent and experienced.

The Team Leader requires 15 years relevant experience, leadership ability, professional qualifications, track record (sic), communication skills and pro-active /co-operative approach.

Subject Matter Experts (SME) must be similarly focussed, albeit 10 years experience in their special field may suffice.

Competency comprises both technical acumen, auditing competence and behavioural competence.

Replacement personnel, if such becomes necessary must be of equal or better capability than the person being replaced.

Relevant skill sets include:

- thorough knowledge of the principles of safety engineering and safety management;
- sound knowledge of AS 61508, EN50126, EN50128, EN50129 and IESM (formerly Yellow Book);
- good understanding of rail based software & hardware systems;
- comfortable in working with upper management, customers and Safety Authorities;
- experience of systems engineering;
- fluent English language skills;
- accredited through a Professional Body;
- ability to think analytically, rigorously and creatively;
- excellent communication skills both written and verbal; and
- degree qualification.

5.2 Scope of ISA services

The ISA services shall cover:

- reviews of safety assurance processes and audits of evidence provided, including independent evaluation of the systems engineering and safety assurance;
    - assurance that the processes are suitable to achieve the objectives and have been implemented effectively;
    - assurance that the evidence demonstrates that safety requirements have been met; and
    - specific reviews and safety audits.
- expert opinion on the overall acceptability of the safety risk; and
- compliance with appropriate standards, international and national good practice.

The ISA will be required to fulfill the following duties:

- plan and resource ISA activities;
- manage the delivery of ISA projects on time and within budget;
- take responsibility for ISA reports within agreed timescales;
- project management activities as appropriate;
- audit contractors safety assurance activities; and
- present assessment findings to the Principal.

The outputs shall confirm the following:

- safety requirements are defined;
- the Works are compliant with all relevant statutory requirements, rules, standards and technical specifications;
- all rail safety risks have been identified and eliminated or reduced SFAIRP;
- the project is capable of being operated and maintained to an acceptable level of safety;
- all necessary safety cases have been produced, assessed and approved; and
- the project is fit for revenue service and public use.

5.3 ISA plan

The ISA plan (ISAP) shall include a description of ISA activities, timelines and deliverables. Compliance to relevant standards is expected for risk management, work health & safety, environmental management and change management,

The structure of the ISAP will define scope, objectives, team, responsibilities, strategy, key performance indicators, procedures, record keeping, outputs and reviews.

The methodology should include: task definition, collection of evidence and risk assessment of the ISA itself.

Assessment techniques include document reviews, safety audit, witnessing, interviews, site inspections, workshops and obtaining expert opinion.

A document management system shall be employed.

5.4 ISA methodology

In reviewing the Project risk methodology, the ISA will itself employ internationally benchmarked risk ranking to identify the most effective types and levels of assessment activities.

Tasks will be defined in terms of activity, type of evidence, tools and techniques, rigour commensurate with level of risk, resources, timeframes and stakeholder interfaces.

5.5 ISA reports

ISA reports shall be evidence-based, informative and actionable. The ISA reports will be:

- internally reviewed;
- list key hazards, issues, findings and recommendations;
- summarise the safety status of the project with reference to relevant benchmarking;
- define unresolved risk and intervention priorities;
- schedule timescales for resolution of outstanding issues.

6 Key points

By way of summary, the paper has provided an introduction to relevant standards – AS 61508 and EN 50126 in particular. These require independent assessment of safety.

Life cycle phases defined by these standards are discussed in terms of reaching an understanding (concept and scope), deriving requirements (hazard and risk analysis and safety integrity levels), certification (realisation and acceptance) and verification and validation (V&V).

A case study was then presented explaining how one of the authors was engaged as the ISA and carried out investigations and recommendations leading to certification and operation of Train Order Working throughout the NSW Country Rail Network. That ISA role is ongoing.

Lastly answers are given to questions asked regarding the ISA:

- The role of an ISA is to investigate and arrive at a judgement as to the achievement of functional safety;
- ISA tasks include planning, gathering evidence and engagement with Stakeholders including Contractors;
- Advantages and disadvantages of ISA are discussed noting that standards now allow multiple routes for providing evidence of compliance;
- ISA shall be qualified, competent and experienced;
- Scope of services and ISA methodology must be planned; and
- Lastly, ISA reports shall be evidence-based, informative and actionable.

The authors thank 4Tel Pty Ltd for permission to present the case study. The experience has provided a strong example of the value of AS 61508 in the development, certification and realisation of safety-related systems.

7 References

Anderson, K (2008) Risk reliability and resilience – Train the trainer. Self published.

AS 61508 (2010) Functional safety of electrical /electronic /programmable electronic safety-related systems.

EN 50126 (1999) Railway applications – Specification and demonstration of reliability, availability, maintainability and safety.

EN 50128 (2011) Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems.

EN 50129 (2003) Railway applications - Communication, signalling and processing systems - Safety-related electronic systems for signalling

Froome, P (2005) Independent safety assessment of safety arguments – Proceedings of thirteenth safety-critical systems symposium, Springer-Verlag.

IESM (2013) International engineering safety management – Good practice handbook.

Utilizing Inherent Diversity in Complex Software Systems

Peter Okech[1] **Nicholas Mc Guire**[2] **Christof Fetzer**[3]

[1] Department of Physics
University of Nairobi,
Nairobi, Kenya,
Email: pokech@uonbi.ac.ke

[2] OpenTech EDV Research GmbH
2193 Bullendorf, Austria,
Email: der.herr@hofr.at

[3] Systems Engineering Group
Technische Universität Dresden,
Dresden, Germany,
Email: christof.fetzer@tu-dresden.de

Abstract

Diversity in simple software systems is known to be of limited effectiveness with respect to systematic faults, whereas in complex systems this picture drastically changes. Diverse implementation of complex software is generally perceived as expensive. In contrast, security uses a different form of diversity to protect against systematic faults: randomization. Explicit memory-space randomization has been shown to be effective for a large class of vulnerabilities. This diversification is carefully designed into the system to assure its effectiveness, but what about the system's inherent non-determinism which manifests itself in inherent diversity of complex systems? Could this randomization be exploited in the form of a diversity argument? We propose an alternative safety strategy utilizing inherent diversity for replicated system architectures to mitigate residual systematic faults. We argue that the effective diversity is a result of inherent non-determinism. Our strategy is to review, analyze and test the dominant execution paths and demonstrate statistical independence of rare paths, allowing architectural protection based on runtime diversity. In this paper we present early results of analyzing the inherent non-determinism of the Linux kernel, and its potential to cover residual faults in the untested execution paths of a complex software system.

Keywords: diversity, complex software systems, randomization, residual faults

1 Introduction

Software based systems are increasingly being called upon to perform safety functions that can no longer be left to electromechanical systems. These software based safety-related systems must be certified as adequately safe in accordance with a generic standard such as IEC 61508 (IEC 2010) or with an industry specific standard before installation and operation. The standards prescribe the set of procedures and techniques to be used in the software development process (e.g part 3 of IEC 61508). In addition, for highly reliable systems, the standards mandate the use of architectural protection to provide coverage for faults during operation of the safety-relevant systems.

A software system typically consists of an application and system software that supports its execution. On one hand, the safety applications that are designed to fulfill the safety function can effectively be designed using the principles of keeping the software simple and safe. In contrast, the system software that forms part of the execution platform is increasingly becoming complex particularly based on the need to manage contemporary hardware resources. We no longer have the luxury of developing all the components in the software stack, but are forced to look at ways of using off-the-shelf complex software components such as general purpose operating systems in safety-related applications.

The use of complex system software in a safety context presents a challenge. The challenge is twofold: first complex systems tend to have a higher density of residual faults and secondly, current prescriptive certification standards are rigid in the approach to demonstrate the safety of the system. A way out of this is to view complexity as a resource that need not be eliminated but exploited. This suggests that we need to look for approaches that enhance safety through the utilization of complexity.

In this paper, we present one such approach to utilizing the non-determinism in complex execution platforms to protect systems against systematic faults in software and suggest a strategy for the demonstration that a system based on this approach is acceptably safe. Specifically, we

- Explain the concept of inherent diversity in a homogeneous multi-channels system, and how it can be used for fault detection in a replicated system architecture (section 4). We argue that using inherent diversity on a 2-out-of-2 (2oo2) system is sufficient to provide safety.
- Use the GNU/Linux kernel as an example of a complex system software that illustrates the variability in path execution. In section 5, we present

The first author gratefully acknowledges support from the Mundus ACP II Programme during the mobility period spent at Technische Universität Dresden where part of this work was done.

our approach to tracing system calls in the Linux kernel and present the results of our experiments.

- Present our hypothesis on the existence of faulty paths in complex software in section 5.4. We then provide a basic argument for justifying the use of inherent diversity (section 6.1) to argue coverage of a subclass of systematic faults in replicated complex software systems.

To investigate the non-determinism in complex software systems, we have used a function tracer to record the functions in the Linux kernel executed by system calls invoked from a test application. Though the system we used was more or less idle, there was variability both in the path and in the execution times of the system calls. We did not consider timing differences in this study but focused on the differences in the kernel execution paths. For a system with a higher load we expect an increase in non-determinism, which is better for inherent diversity.

The trace data of each system call execution was analyzed. Several paths for a given system were identified, with some paths being more frequently taken than others. It can be assumed that the rare paths are not well tested and thus any residual faults in those paths would stay undetected. Given that these paths are taken in very rare circumstances, the probability of any applications in any two channels running concurrently taking the same path is sufficiently low.

Further, we claim that taking one of the rare paths within the kernel is a random event based on the inherent non-determinism of the system state. If this claim holds, then we can protect the system against systematic faults in these rare path by architectural means such as a 2oo2 system in the same way the architecture will provide protection against true random events like bit flips. This provides the foundation of the argument that we propose for justifying safety based on inherent diversity.

We believe that it is reasonable to make the above claim of concepts based on the data from the work reported in this paper. However, these are our early results and definitely more work still needs to be done to arrive at a conclusive assessment of inherent diversity.

2 Problem Definition

Consider a software engineer tasked with the problem of developing a software based safety application today. The system will need to be developed according to the prescribed procedures and processes and, be certified as safe by an appropriate safety standard. Suppose the safety standard provides the possibility of using contemporary hardware (with performance enhancing features, multicore, hardware acceleration etc) for which the required selection process is duly followed. The main issues the engineer has to deal with would then be on how to realize the software that performs the required safety functions. Naturally, the software will consist of the safety application and system software for example, an operating system.

Can the engineer make use of a general purpose operating system, such as GNU/Linux that was not specifically developed with safety concerns during its design or implementation? The use of such complex systems would pose two challenges:

- There is an expectation that such a system will contains a high number of residual systematic faults.
- How would the system be justified as adequately safe, given that it was not developed using a process compliant with a safety standard.

We suggest the use of inherent diversity as the architectural means to provide coverage of residual systematic faults in the complex software system and propose an approach to argue the safety of a system based on inherent diversity.

3 Background

Before discussing the approach to inherent diversity and the process of quantification of safety, we provide some background in this section.

3.1 Software Diversity

Dependable systems incorporate fault tolerance features to first detect errors, and then to mask, avoid or recover from the effect of these errors. To achieve these mitigation, redundancy techniques are typically used to implement fault tolerance; hardware redundancy to provide coverage for random faults and diversity for design faults. A central theme in this section is how to generate diversity in software systems. We first provide a brief overview of redundant architecture and how diversity is used in software based systems for the purpose of achieving safety using some adjudication scheme.

3.1.1 Redundant Architecture

The use of redundant channels in safety-related systems predates programmable electronic systems. There are several configurations or architectural protection schemes that can be used to cover random faults, or at least detect such faults. These configurations are generally referred to as M-out-of-N (MooN) systems. The simplest redundant system is the 1oo2 in which one channel is enough to perform its function. The configuration is used to achieve availability, e.g. in hot standby mode, but never used in safety. The 2oo2 system requires both channels to perform the safety function which allows for a single fault detection but provides no fault masking. The 2oo3 or the triple modular redundancy (TMR) provides single fault coverage, masking the fault in one of the three channels enabling it to perform its safety function with two healthy channels as well as providing high reliability. Other configurations require more resources and are rarely used in practice. In this work, we will use the 2oo2 system as an example to illustrate concepts.

3.1.2 Design Diversity

Due to the acceptance of the notion that all software faults are design faults, it has generally been perceived that pure replication of software would result in the channels having the same faults, and thus will result in a common cause failure (CCF). Randell (1975) proposed design diversity, the use of a spare component whose design was independent from the main components, to protect redundant systems from CCFs. A widely used technique for generating diverse versions of software components is N-version programming (Avizienis & Chen 1977). The idea is that N different teams are provided with the same specification to develop different implementations (versions) of the software component. A further modification to the concept is to force the developers to use different languages and/or methodologies (Littlewood 1996). Given the same input, the different versions run concurrently and their output is compared using some adjudication mechanism.

The basic idea of design diversity is that the versions running will fail independently, thus exhibiting failure diversity. Assuming independence between failures of the channels, it is possible to claim much higher reliability of the system than that of the individual channels. Both empirical analysis (Knight & Leveson 1986) and theoretical work (Eckhardt & Lee 1985, Littlewood & Miller 1989) show that even for independently developed software versions, there are positive correlation between channel failures. This is attributed to the fact that for given programs, failures are more likely to happen on certain demand than others.

Though design diversity is an effective way of improving the dependability of software based systems (Littlewood et al. 2001), its uptake has been low except for very high integrity systems. The main reasons are its cost effectiveness and well as the inability to correctly quantify the gains in dependability of systems based on design diversity.

The automatic generation of diverse versions has been proposed as a means of reducing the high cost involved in manual development, testing and maintenance of different versions. The common approach is to use a compiler to generate the diverse copies from the same base program code. The technique has been used to tolerate hardware faults (Gaiswinkler & Gerstinger 2009) and is also increasingly being used in other dependability contexts such as security.

3.1.3 Diversity in the Security Domain

The security community recognizes that design faults can be taken advantage of by malicious users, thus compromising systems. To protect against these incidences, the ability to use the known faults also termed as vulnerabilities, is made as expensive as possible through the use of randomization. By randomizing features such as memory addresses and/or instructions, a large number of vulnerabilities can be effectively dealt with. To ensure its effectiveness, this randomization is explicitly designed into the system.

Unlike the safety domain which focuses on the the use of specifically designed and verified software stack, the security domain has for a long time accepted the use of general purpose operating systems for their applications. A key aspect is the recognition that it is not possible to eliminate all systematic faults from complex software systems by methodological life-cycle development only. Rather than eliminating individual defects in such systems, the broad approach used by the community has been to obscure these faults by introducing non-determinism in the execution environment, thus deriving diverse copies of systems.

The main lesson from the security community applicable to our work is that there exists a generalizable strategy to mitigate systematic faults: through randomization of an execution environment, systematic faults can effectively manifest themselves as random faults. Thus, the well established mitigation strategies for random faults can be applied to mitigate randomized systematic faults.

4 Inherent Diversity

A complex system always displays behavior that arises from the interaction of its components and such behavior cannot be identified by looking at the components in isolation. Modern hardware/software systems are built from processors that exhibit a certain level of inherent randomness associated with complexity rather than particular code paths (Mc Guire et al. 2009). This inherent randomness is amplified by complex operating systems.

Computing platforms thus have a reasonable amount of inherent diversity. This is a property of a system that for a given input vector, the system will not have a deterministic path of execution. In this work, we describe a path as a sequence of functions or routines that are called during an execution, neglecting execution time differences. In the context of kernel path variability, a path is the set of kernel routines called during the execution of a system call when a user application requests a service by invoking the system call.

Inherent diversity arises due to several factors at play in a complex system. First, these systems have a very large state space, i.e. many variables and other data structures. Due to this it is conceivable that during the lifetime of the system, some of the states are never re-visited. Secondly, most complex systems are concurrent systems and the parallelism intrinsic in these systems increases the potential for non-determinism. The third factor contributing to the diversity is the presence of a large number of asynchronous events. Together, these factors lead to unpredictable internal states of of the system and thus non-deterministic paths from input to output, without impacting the correctness of a task.

Consider for example a 2oo2 system with replicated non-diverse safety application. We make the assumption that safety application is simple, and thus can easily be verified using existing tools while the operating system is complex and contains residual faults. Due to the inherent randomness of the execution platform, the two channels will diverge in state (see Figure 1). If each of the applications invokes a system call, the execution path in the kernel will likely take different paths. Suppose there is an untested path containing a fault, the probability of two systems taking the same erroneous path at the same time is much less than one of them taking this path. Using adjudication, it is possible to detect the fault. We believe that the 2oo2 architectural configuration based on inherent diversity is sufficient to provide safety even for highly critical systems.

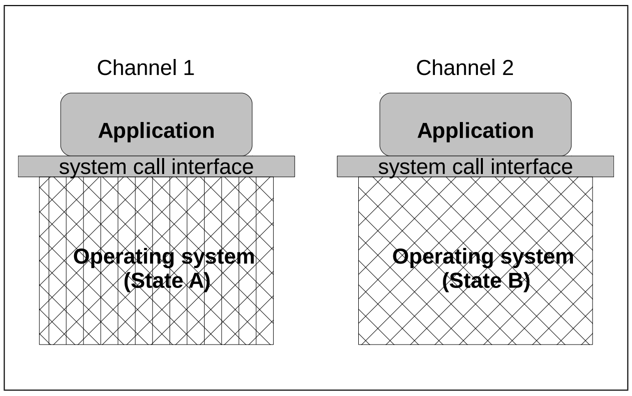


Figure 1: Diversity in a redundant channel

5 Evaluation of Non-determinism in the Linux kernel

To support the claim of inherent non-determinism which manifest as inherent diversity in complex software systems, we set to investigate this phenomena in a real world complex system: the GNU/Linux operating system.

We set up an experiment to trace the execution of system calls in kernel space and performed a top level analysis of the data collected. In this section, we describe our experimental set up, method and the results obtained. We then complete the section by a discussion of the results that were obtained.

5.1 Experimental Environment

To run our experiment, we used a generic PC, with the specification typical of machines available to the community for use in control systems. The hardware/software specifications of the experimental platform were as follow:

- Intel Core 2 Duo CPU running at 3.00 GHz with 1GB main memory
- GNU/Linux (Debian release 6.0), Kernel version 3.12.0 #SMP PREEMPT
- GCC 4.4.5

Our interest was the analysis of execution events associated with an application in kernel space. To support kernel tracing, the utility Ftrace (Rostedt 2009) was enabled in the kernel configuration.

5.2 Experiment Description

The first step of the experiment was the development of a user application that would invoke systems calls to request for operating system services. We make use of a simple example that reads from a value from a binary file, increments the value and then writes back the new value into the file. The read-write cycle is performed within a loop. The application's pseudocode shown below.

```
BEGIN
    open file
    for count from startValue down to 0
         read data from file
         increment data
         write data to file
         sleep for 10 sec
    close file
END
```

The size of the program's loop was set to 20,100 iterations. After compiling the test program with gcc defaults, the program was launched immediately after a fresh boot of GNU/Linux in a command line mode. This provided a more or less idle system with minimal services executing in the background.

We used trace-cmd (Rostedt 2010), a front-end tool to Ftrace, to record the traces of the kernel functions called by the test program. The output of this is a text file containing the call graphs of each of the system calls made by the test program. To aid the analysis task, the trace file was pre-processed using shell scripts in order to convert the file to text delimited format suitable for exporting/importing to a PostgreSQL database. We chose to transfer the data to a database to make it easy to perform complex queries and analysis on the data.

5.3 Results

In the execution run, a total of 180,975 system calls were recorded in the trace file. Most of these were the repetitive invocations of system calls in the set {*read*, *write*, *lseek*, *fsync*, *nanosleep*}. These system calls were explicitly invoked by the test program. We then performed a two level analysis on the data collected, one on individual system calls, and the other on the set of system calls in the program iterative loop.

As described in section 4, an execution path of a system call is a sequence of kernel functions that are called during that system call. The length is thus the number of functions in this sequence. The number of functions called during an execution instance depends on both the system call as well as the execution environment. The variability in the duration of execution and the length of the paths for system calls in the set {*read*, *write*, *lseek*, *fsync*, *nanosleep*} is shown in Table 1. The large difference in both timing and path lengths are due to among other factors, preemption of the system calls during execution.

	Duration in us		length of path	
	min	max	min	max
read	15.279	23253.259	77	692
write	8.836	407.395	12	582
lseek	1.915	144.872	4	225
fsync	8418.372	753850.994	402	1154
nanosleep	9990266	9990638	47	583

Table 1: Timing and Path Variability of the data set

Since our focus is on the paths taken by individual system calls, we present in Table 2 a summary of the path characteristics for the system calls in the test program's iterative loop.

	Number of		Most Common Path	
	Calls	Paths	Freq.	%
read	20201	636	6057	30.13%
write	20132	768	11748	58.35%
lseek	40200	33	40164	99.91%
fsync	20100	16139	36	0.18%
nanosleep	20100	174	14189	70.59%
rt_sigaction	20100	22	20078	99.89%
rt_sigprocmask	40200	81	40100	99.75%

Table 2: Path Characteristics of the data set

Looking at the iterative loop of our test program, we identified 19,505 unique paths (i.e. the combine paths of the system calls in the body of the program's loop), with the occurrence frequencies as depicted in Table 3.

Freqency	8	7	6	5	4	3	2	1
Instances	1	1	9	5	17	61	342	19069

Table 3: Paths in the Program's Iterative Loop

5.4 Discussion

In this section, we discuss the results presented above, and provide an interpretation in the context of IEC 61508, with respect to the identification of possible failure mechanism in software.

Since we are addressing a generic software component for the use in a safety-related system we do not define the specific safety function of the component. Rather, we assume that a safety application compliant to IEC 61508 can call upon a particular function in the operating system which, if faulty, has the potential to seed a fault. A conservative approach is thus taken in this work, and each residual fault needs to be considered safety related. As it is difficult to identify each and every fault in a complex software component we seek a generic rather than a specific mitigation.

5.4.1 Variability in the execution path

The experimental results achieved show that there is both temporal and path variability of a system call from repeated execution of a program. Of interest to our work is the variability in the execution path.

In a large set of runs, we can identify several execution path through the kernel space. From the point of view of the invoking application, these are functionally equivalent paths, but internally and transparent to the application, they exhibit diversity. We can attribute the path diversity to the internal state of the operating system, which is different every time a system call is invoked.

For the *read* system call, there are a total of 636 distinct paths from the 20,101 system call instances. 591 or 92.92% of the path were taken by only one system call instance. This represents only 2.94% of the total number of system calls considered, which represents a small fraction of the total number of calls. These unique instances mainly consist of those system calls that were interrupted by asynchronous events (Okech et al. 2013). We generally expect that with a high system load, the proportion of these paths would increase. The remaining 19,510 instances took one of the other 45 execution paths. We can view these paths as being in two groups, frequently taken paths and rare paths.

Consider the most frequent path taken in this execution run. It has a frequency of 6,057, meaning that 30.13% of the system calls did take this path. If we consider the two most common paths, we find that these paths were taken by 10,968 system call instances, representing 54.56% of the total. Similarly the top three and four paths are taken by 13,675 (68.03%) and 15,917 (79.19%) respectively. The top ten most frequently taken paths are 92.32% of the total read system calls. We present the cumulative frequencies of the top 14 paths of the read system calls in Table 4. Not shown in the table are paths (1007) with a frequency of less that 95.

The results obtained gives us confidence that we can classify paths into two groups - frequently taken paths and rare paths - for any non-trivial system call. From our experiment, the *lseek* system call illustrates the properties of what can be referred to as trivial system call. Out of the 40,200 instances, 40,164 of these take the same path (see Table 2). Due to the low variance in its paths, the *lseek* system call can be well tested.

We contend that with a conservative test coverage of about 80% for non-safety critical systems, a test campaign on complex systems within a reasonable budget will only exhaustively test the most commonly taken paths. We believe that it is not feasible using current techniques to write test cases to exercise all possible paths in a complex system such as the Linux kernel. This situation leads to a very high possibility that the rare paths are not reasonably well tested, and hence will contain residual systematic faults.

Top N	Number of Paths		Cumulative Number	
paths	No.	percentage	No.	percentage
1	6057	30.13%	6057	30.13%
2	4911	24.43%	10968	54.56%
3	2714	13.50%	13682	68.07%
4	2242	11.15%	15924	79.22%
5	1018	5.06%	16942	84.28%
6	466	2.32%	17408	86.60%
7	450	2.24%	17858	88.84%
8	306	1.52%	18164	90.36%
9	211	1.05%	18375	91.41%
10	182	0.91%	18557	92.32%
11	152	0.76%	18709	93.07%
12	148	0.74%	18857	93.81%
13	142	0.71%	18999	94.52%
14	95	0.47%	19094	94.99%

Table 4: Path Frequency Table

IEC 61508 (2010) notes that diversity is one of the procedural methods available for mitigating against systematic faults. Inherent diversity as presented in this paper is attractive for addressing this issue as it allows arguing a generic mitigation of residual systematic faults, that, given the overall complexity of the Linux kernel could not be covered analytically or by testing with economically tolerable effort.

5.4.2 Correlation of paths

The main premise of our work to assume independence in the taking of the paths in the rare paths set. Therefore an important issue is to find out whether the kernel paths taken by a system call execution displays an identifiable patterns over time. In this section, our analysis focuses on the *read* system call only.

We first looked at the relationship between the system call instances and the path taken. A snapshot of the plot of the instances against the index of the path taken is shown in Figure 2.

To find out if the next path taken by a system call instance is dependent on the previous paths i.e. are the path taken conditioned on path history?, we performed tests on the data using the Autocorrelation Function (ACF).

The autocorrelation coefficient of a series of values lies in the range [+1:-1] with values closer to these limits signifying strong correlation, and therefore indicating departures from the assumption of independence. If a series is uncorrelated, then the expectation is that the autocorrelation coefficient will be close to 0. The plot of the autocorrelation coefficients for the *read* system call is shown in Figure 3.

In our data, the highest coefficient has a magnitude of 0.1536 with the values becoming much smaller as the lags gets larger. Additionally, only 403 (or 8.02%) coefficients fall outside the 95% confidence interval. Due to the low magnitudes of the autocorrelation coefficients, we can safely make the assumption that the paths are independent.

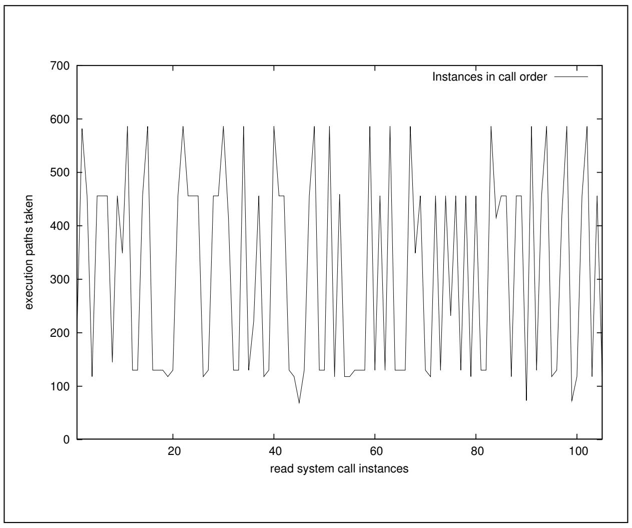


Figure 2: Plot of instances (in call order) against path taken

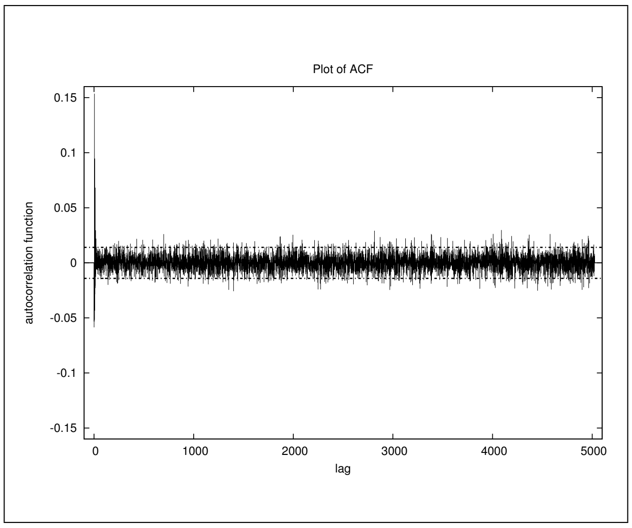


Figure 3: Autocorrelation of the *read* system call with 95% confidence interval (dashed line)

The key issue for us is to show statistical independence in an execution taking one of the rare paths. At the current point we are in our work, we have not yet identified a suitable method to isolate the rare path set, for which tests for independence can be performed.

6 Safety Justification based on Inherent Diversity

Developers of of software based safety-related systems have several available routes to certification of their products. The first and the one with with the highest chance of success is following the prescribed process; specify requirements, develop, verify and operate. A second option is provide a proven-in-use argument for the software incorporated in the safety system if the component has not been developed from scratch. There exists a third option if a pre-existing software is used. In this later case, the focus is on the selection process and assessment of available life-cycle data.

The option to select off-the-shelf software has many challenges. There are several issues to be considered if and when a developer decides to use software designed and developed without having functional safety as one of the primary goals. However, there are alternative strategies that could be used to demonstrate safety of such systems.

We suggest an approach justifying safety-related systems based on the inherent diversity. There are two aspects of the strategy:

- Safety argument
- Procedure for assessment of safety

6.1 Safety Argument

The first issue to consider is the validation process of the GNU/Linux kernel. In commercial software development, detailed specification are made and then used to perform validation of the final product. On the other hand, in the Linux development model, the development process and the traceability are informal. Though the kernel is developed in a controlled manner by a high quality development team with a well thought-out development and maintenance process, no (semi-)formal specification of the kernel is available.

One of the most important aims of Linux is the compliance to the POSIX (Institute of Electrical Electronics Engineers 1988) specification. There exists a semi-formal specification in the form of the POSIX interface specification which can be used to validate the kernel. This is already being done by the Linux Test Project (LTP) (LTP 2013). The project provides a test harness that can be used in a systematic way to run the POSIX test suite against the Linux kernel, thus validating the kernel against the specification to demonstrate that it meets its stated requirements.

However, no amount of testing will result in a complete coverage of a complex system such as the GNU/Linux kernel. It is infeasible to write test cases that will cover the entire possibility of execution paths in the kernel. There is a potential that some residual faults remain in those paths that have not been exhaustively tested. We need to provide protection against these residual systematic faults. We suggest a mitigation though combination of architectural means and inherent diversity.

Safety Risk There is a safety risk associated with the residual faults that remain in the operating system. This could take the following form:

> a correct and valid input vector from an application submitted to the operating system through the POSIX interface invokes a system call, but the request made took an execution path that was not covered by the tests that were done during the validation process.

We take the position that for some paths a test case could not be provided, thus these paths belong to the rarely taken path set. Further, we know that an application's input vector cannot deterministically cause a rare path to be taken. If this was possible, then the it would have been covered by a test case.

Assuming that taking the rare path is a statistically independent event then we are essentially taking residual systematic faults in complex software and transforming the effective behavior of that fault on the system to appear as a random fault based on inherent non-determinism. This transformation can only be possible if we design an appropriate architecture for the overall system, say as a minimum a 2oo2 system.

The probability of both channels entering the same faulty rare path at the same time is sufficiently low. The residual faults can be regarded as random faults and is sufficiently protected by architectural means of a 2oo2 system. The claim is that if the taking of a rare path is a random event based on the inherent diversity of the system, then the systematic faults in these paths can protected by architectural mechanism in the same way the architecture protects against truly random faults.

The significant threat to the correctness of the system is if the residual faults present in the untested paths are not statistically independent. In this case, these represent common cause faults, which would require further analysis.

The overall approach we use to justifying safety is illustrated by the diagram of Figure 4.

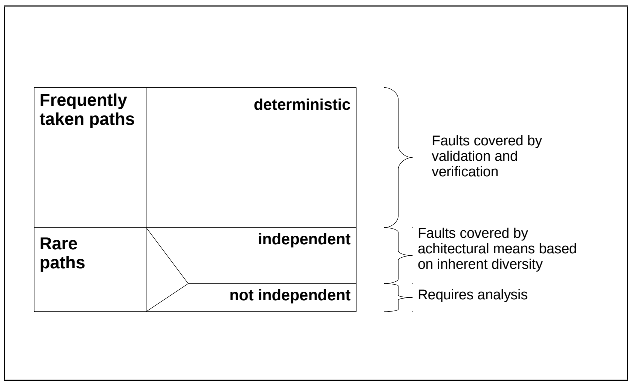

Frequently taken paths
deterministic
Faults covered by validation and verification
Rare paths
independent
Faults covered by achitectural means based on inherent diversity
not independent
Requires analysis

Figure 4: Summary of the safety strategy

6.2 Procedural Perspective

The counterpart to the justification of safety above is the approach to quantification of safety levels achievable through this mechanism. The main goal of the approach is to establish a firm process with sufficient level of assurance so that we can quantify the independence of the paths. This has to be by a reasonable reproducible and sufficiently assured process. This quantification can then be used to do an assessment of a 2oo2 system based on inherent diversity. This is the focus of our ongoing work.

7 Related Work

With respect to the method used to show the diversity in path execution, the SIL4Linux project (Wang et al. 2009) is closest to our work. While one of their main aims was to check the real-time behavior of the Linux kernel in the context of safety, we are more interested in determining the variability in the behavior of specific system calls during its execution.

Similar to the goal of our work is one of the stated research objectives of the INDEXYS project (Eckel et al. 2010) to investigate how inherent diversity of complex operating systems helps in detecting faults in computing platforms. The project targets to instantiate platforms such as the TAS Control Platform (Gerstinger et al. 2008), which employs architectural protection schemes to mask and/detect random faults through temporal relaxation. The loose coupling is based on the assumption that the channels are temporally independent so that any fault activation in one of the channel is detectable. This is building on the non-deterministic property of loosely coupled systems, though our work focuses on the path non-determinism in addition the non-determinism in the temporal dimension.

8 Conclusion

We believe that functional safety should be based on confidence and assurance not on prescribed procedures and processes. Technology in the area of programmable electronics has been undergoing fundamental changes in the past decade. At the same time the complexity of system software that support execution of user applications is growing. These changes at the "foundations" of safety-related systems will need to be addressed by the safety community. It is our conviction that diversity in the software stack is a key methodology in tackling the challenges of ensuring the dependability of safety related systems.

Though the core concepts of diversity are established in safety engineering, the traditional approaches to achieving diversity are reaching their limits, both technically and economically. It is on this premise that we propose the concept of inherent diversity. Establishing inherent diversity as a sound methodology for safety-related systems will need considerable more than the proof-of-concept level we have so far engaged in. The inherent diversity approach presented in this paper is by no means a stand-alone argument that can resolve the assurance demands of safety-related systems verification utilizing complex hardware/software but it is, in our opinion, a radically new approach to diversity that shows a promising potential to play a role in enabling complex software, like the GNU/Linux operating system, for safety related systems.

References

Avizienis, A. & Chen, L. (1977), On the implementation of n-version programming for software fault tolerance during execution, *in* 'Proc. IEEE COMPSAC', Vol. 77, pp. 149–155.

Eckel, A., Milbredt, P., Al-Ars, Z., Schneele, S., Vermeulen, B., Csertn, G., Scheerer, C., Suri, N., Khelil, A., Fohler, G. & et al. (2010), Indexys, a logical step beyond genesys., *in* E. Schoitsch, ed., 'SAFECOMP', Vol. 6351 of *Lecture Notes in Computer Science*, Springer, pp. 431–451.

Eckhardt, D.E., J. & Lee, L. D. (1985), 'A theoretical basis for the analysis of multiversion software subject to coincident errors', *Software Engineering, IEEE Transactions on* **SE-11**(12), 1511–1517.

Gaiswinkler, G. & Gerstinger, A. (2009), Automated software diversity for hardware fault detection, *in* 'Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on', pp. 1–7.

Gerstinger, A., Kantz, H. & Scherer, C. (2008), Tas control platform: A platform for safety-critical application, *in* 'ERCIM', ERCIM.

IEC (2010), '61508 functional safety of electrical/electronic/programmable electronic safety-related systems', *International electrotechnical commission* .

Institute of Electrical Electronics Engineers (1988), Portable operating system interface for computer environments, Technical Report 1003.1-1988, IEEE, New York.

Knight, J. & Leveson, N. (1986), ‘An experimental evaluation of the assumption of independence in multiversion programming’, *Software Engineering, IEEE Transactions on* **SE-12**(1), 96–109.

Littlewood, B. (1996), ‘The impact of diversity upon common mode failures’, *Reliability Engineering & System Safety* **51**(1), 101 – 113.

Littlewood, B. & Miller, D. R. (1989), ‘Conceptual modeling of coincident failures in multiversion software’, *Software Engineering, IEEE Transactions on* **15**(12), 1596–1614.

Littlewood, B., Popov, P. & Strigini, L. (2001), ‘Modeling software design diversity: A review’, *ACM Comput. Surv.* **33**(2), 177–208.

LTP (2013), ‘Linux test project home page’, `https://sourceforge.net/projects/ltp.` (assessed on 2014-01-28).

Mc Guire, N., Okech, P. & Schiesser, G. (2009), Analysis of inherent randomness of the linux kernel, *in* ‘Proc. of the 11th Real-Time Linux Workshop Workshop, Dresden’, OSADL.

Okech, P., McGuire, N., Okelo-Odongo, W. & Fetzer, N. (2013), Investigating execution path non-determinism in the linux kernel, *in* ‘Proc. of the 15th Real-Time Linux Workshop Workshop, Lugano-Manno’, OSADL, pp. 180–186.

Randell, B. (1975), ‘System structure for software fault tolerance’, *Software Engineering, IEEE Transactions on* **SE-1**(2), 220–232.

Rostedt, S. (2009), ‘The World of Ftrace’. Linux Foundation Collaboration Summit.

Rostedt, S. (2010), ‘trace-cmd: A front-end for Ftrace’, `git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git`. (assessed on 2014-01-28).

Wang, L., Zhang, C., Wu, Z., Mc Guire, N. & Zhou, Q. (2009), Sil4linux: An attempt to explore linux satisfying sil4 in some restrictive conditions, *in* ‘Proc. of the 11th Real-Time Linux Workshop Workshop, Dresden’, OSADL.

Operational Hazard Analysis in DoDAF

MURRAY BAILES
School of Information Technology & Electrical Engineering,
The University of Queensland, Brisbane, Australia
Functional Directions Pty Ltd
56 Fairlight Street, Fairlight, NSW, 2094
murray@functionaldirections.com.au

Abstract

A method is proposed for performing an Operational Hazard Analysis (OHA) that employs a HAZOPS style analysis technique performed on the operational models of a DoDAF[1] Model Based Systems Engineering (MBSE) capability architecture framework to identify the implementation agnostic safety requirements for the capability.

OHA commences by examining the safety risk associated with each Operational Activity (OA) in the context of the emergent behaviour of the overall operational model. The analysis considers both the potential consequence of the failure modes of each OA as well as any role that the OAs may play in reducing the overall system level safety risk.

Based on the OHA, a set of Preventative Controls and Systems Safety Requirements are derived and traced to the elements of the capability architecture for implementation and verification.

The OHA technique is demonstrated on a model of a hypothetical Aerial Firefighting Management System (AFMS).[2]

Keywords:
DoDAF, Model Based Systems Engineering, MBSE Functional Hazard Analysis, FHA, HAZOPS, Operational Hazard Analysis, OHA.

1 Introduction

Requirements (and constraints), and requirements models under the MBSE paradigm, provide the essential language of Systems Engineering.

User or operational requirements specify the needs for which a system is being developed to meet; Technical or system requirements specify the functional and physical characteristics of the implementation; and statements of work requirements specify the system development process and its deliverables.

Respectively, these requirement types represent the: 'What do we want?' (Operational View); 'How is it implemented?' (System View); and 'How do we build, operate and maintain it?' (Program View); information perspectives that collectively define a Capability Architecture. (Defence Capability Development Manual 2006).

Figure 1 conceptualises these three primary information views within the overall capability architecture. This paper proposes a hazard analysis technique that specialises the program activities of the CA. The technique utilises the DoDAF relationships between the elements of the operational and system views to support a hazard analysis that can traverse the operational to system boundary.

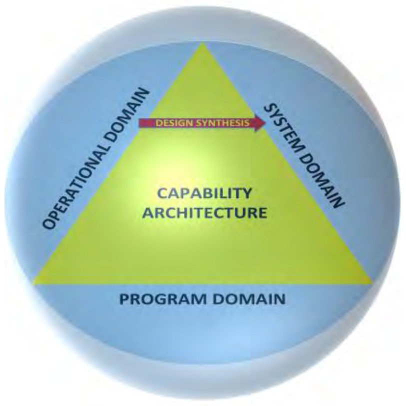


Figure 1 Capability Architecture Information Views

The models presented in this paper were constructed using CORE™ utilising the DoDAF 2.0 schema which was supplemented with additional information classes required to support OHA. CORE was also used to provide the MBSE Design Repository (DR) to capture the design rationale for the capability as well as to provide the underlying model building and requirements traceability.

Utilising the mainstream MBSE architecture to perform a hazard analysis enables the safety engineer to utilise the top down MBSE models to provide a user centric view of the capability that can be shared by both

[1] The capability architecture models presented in this paper were constructed using the CORE DoDAF 2 schema. CORE is a MBSE Design Repository produced by the Vitech Corporation. www.vitechcorp.com

[2] Copyright © 2014, Australian Computer Society, Inc. This paper appeared at the Australian System Safety Conference (ASSC 2014), held in Melbourne 28-30 May, 2014. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 156, Ed. Tony Cant. Reproduction for academic, not-for profit purposes permitted provided this text is included.

the hazard analysis and mainstream systems engineering processes.

"Without understanding the purpose, goals, and decision criteria used to construct and operate systems, it is not possible to completely perform and understand the hazard assessments that underpin the safety engineering processes required to build safe systems that effectively minimise, and hopefully prevent accidents. This applies not only in the initial system design but also as the system evolves and changes over time." (Leveson, 2004).

2 Innovation and Importance

The innovation within this paper stems from three sources.

1. OHA provides a means to leverage the operational views of commonly used architecture frameworks, such as DoDAF, to support an implementation-agnostic hazard analysis that can utilise the traceability between the operational and system views to create a risk assessment and treatment framework that can support an impact assessment that can traverse the operational to system boundary.
2. The proposed method is intended to utilise the capabilities of commercially available MBSE toolsets to integrate the hazard analysis and Systems Safety Requirements (SSR) derivation and management with the use of a 'living' MBSE DR to realise a 'living' Safety Management System to support the full capability lifecycle.
3. Since OHA is based on the implementation agnostic operational views of a capability the outputs produced provide a template for a generic capability 'type' safety assessment.

3 Functional Hazard Analysis

In their paper, Deriving Safety Requirements Using Scenarios, Allenby and Kelly, (2001) identified problems that are commonly encountered when performing a traditional Functional Hazard Analysis (FHA) to include:

- A poor functional requirements definition;
- A lack of an adequate functional representation to enable consideration of system state, sequencing of functions and dependencies between components;
- The inability to assure completeness of the FHA;
- Difficulties integrating the safety related requirements derived during the assessment into the main system requirements set.

Integrating Safety Engineering with the use of standardised architectural frameworks and a MBSE DR provides a mechanism to overcome many of the problems identified by Kelly and Allenby. (NDIA 2011)

The development and use of a set of architectural models to provide a living baseline of a capability that can be shared by both the system design team and speciality engineering activities such as safety promotes communication within the design team (NDIA.2011).

Integrating the definition, management and verification of the systems safety requirements with the mainstream requirements management processes to create a single point of reference for the system verification and validation activity is expected to reduce duplication of effort and realise efficiencies within the overall design activities.

4 Operational Hazard Analysis

The OHA technique proposed in this paper extends the traditional HAZOPS approach of considering the failure modes of an existing system or working design to consider the failure modes of each of the OAs within the overall operational context described by the Operational Views of DoDAF style capability architecture. It also seeks to identify the role of each OA in reducing the systems level safety risk within the overall operational context.

OHA utilises the Operational Views of capability architecture, that are developed during the concept definition phase, to support an informed preliminary hazard analysis that when performed at the initial stages of a capability development can identify the implementation–agnostic SSR for the capability and guide the subsequent design decisions.

Undertaking an OHA helps to understand the role of each OA in the context of the emergent functionality within the top level OA.

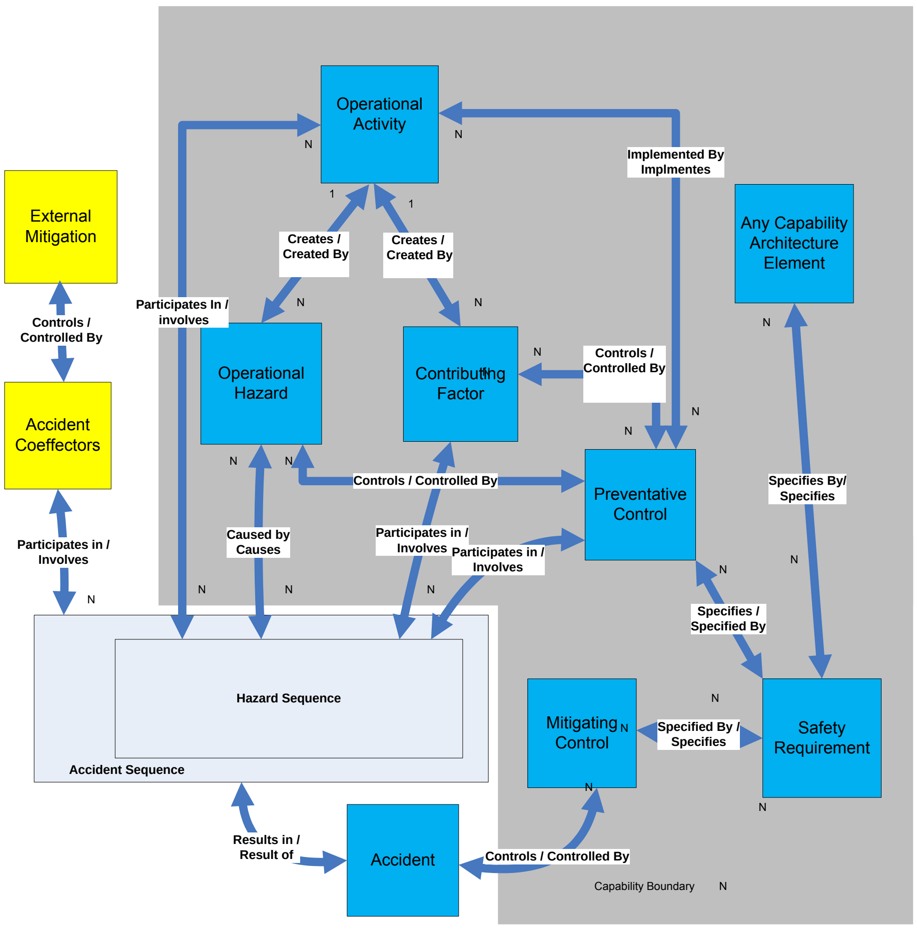


Figure 2 Hazard Management Information Classes

OHA considers the potential of each OA to:

- Create an Operational Hazard. If the failure of the OA creates a safety risk on the capability boundary. An Operational Hazard is the last event or state in a Hazard Sequence that, possibly with the participation of one or more External Co-effectors, can directly result in an Accident
- Create Contributing Factor, if the failure of the AO can participate in the Hazard Sequence for an Operational Hazard. A Contributing Factor cannot directly result in an Accident, but rather can participate in the Hazard Sequence that causes an Operational Hazard that may in turn result in an Accident.
- Implement a Preventative Control if the OA controls the likelihood of an Operational Hazard or Contributing Factor participating in a Hazard Sequence.

Utilising the logic defined within the Operational Activity decomposition a Hazard Sequence describes an ordered combination of Contributing Factors that can cause an Operational Hazard. Hazard Sequences describe the potential role of Contributing Factors to participate in an Operational Hazard.

Accident Sequences extend Hazard Sequences to show how a Hazard Sequence, concurrent with one or more External Co-effectors, can result in an Accident.

Preventative Controls can be added to the Hazard sequences to reduce the likelihood of the Contributing Factors and subsequent Operational Hazard.

External Co-effectors are events or states that are external to the capability that may, in conjunction with an Operational Hazard result in an Accident.

Based on the identified set of Operational Hazards, Contributing Factors, External Co-effectors and Accidents:

- Additional Preventative Controls may be defined to eliminate or reduce the likelihood of each Contributing Factor or Operational Hazard.
- External Mitigations may be added to Accident Sequences to reduce the likelihood of the External Co-effectors participating in an Accident.
- Mitigating Controls may be defined to reduce the impact or consequence of an Accident.

The Preventative and Mitigating Controls become the foundation of the SSRs. Within the DR SSRs can be traced to any element of the CA, and managed using the same requirements management processes used to manage other system level requirements.

Within the CA a SSR may be traced to:

- An element of the Operational View, if that element implements the SSR e.g. an OA implements a defined Preventative or Mitigating Control.

- An element of the Systems View if a Component, Function or interface implements the SSR.
- An element of the Program View if that SSR calls for a particular rigour in some aspect of the design and build processes.

5 Related Work

Similar techniques to OHA have been proposed in the past, such as:

a. Systems Theoretic Accident Model and Processes (STAMP) (Leveson, 2004);
b. A method for deriving safety requirements using use case scenarios (Kelly and Allenby, 2001); and
c. Model-based safety risk assessment using Behavior Trees (Lindsay 2012).

STAMP employs system theory to analyse accidents, particularly system accidents. Behind the STAMP methodology is the concept that accidents occur from control failures when external disturbances, component failures, or dysfunctional interactions among system components are not adequately handled by the control system. Leveson has further proposed STAMP Based Hazard Analysis (STPA) based on the STAMP model of accident causation

Within STAMP safety is viewed as a control problem with layers of control within hierarchical structures containing iterations, feedback loops and information and control flow (Leveson 2004). Leveson also proposed the use of Intent Specifications (Leveson 2000) to model the control problem, support the safety assessment and produce a software specification. Safety Based Systems Engineering Model Based Systems Engineering – Part 1 (Herring et al, 2007) combined STAMP, Intent Specifications and STPA integrated with state analysis performed[3] to provide a holistic approach to hazard analysis.

DoDAF style architecture frameworks provide an alternative method of modelling and analysing hierarchal structures, that contain feedback loops and information and control flow that is more widely used within the mainstream systems engineering community. While providing a complete mapping of the layers of an intent specification to those of a DoDAF capability architecture is beyond the scope of this paper it could be considered that the DoDAF operational model is similar to the system purpose layer of an Intent Specification and that the interface between the system purpose layer and the system design layer of an Intent Specification is similar to the interface between the systems and the operational views within DoDAF.

Deriving Safety Requirements Using Scenarios (Kelly and Allenby, 2001) proposed an integrated approach to requirements expression and failure identification where both activities share a single requirements model of sufficient expressiveness to ensure the clarity and conciseness of the requirements. They suggested the adoption of a model based around operational use case specifications however they cited problems associated with the lack of an adequate functional baseline and poor integration within mainstream design processes.

OHA utilises the operational views of a capability to provide a capability description on which to perform an initial hazard analysis while the defined DoDAF traceability, from the operational to the system viewpoint within the DR, allows integration of the hazard analysis within the mainstream Systems Engineering traceability and requirements management processes to provide the means to address the issues identified by Kelly and Allenby.

The use of Behavior Trees (BT) shares common objectives with OHA however the BT approach requires the development of a separate model for the safety analysis. This limits potential integration of the safety engineering with the mainstream systems engineering artefacts.

6 The Aerial Firefighting Management System Example

This study of OHA uses an architectural model of a hypothetical Aerial Firefighting Management System (AFMS). The model was developed from a set of 56 requirements that were previously used to demonstrate Model-based safety risk assessment using Behavior Trees, (Lindsay et al, 2012). Based on these 56 requirements the operational view of the AFMS was developed for the demonstration of the OHA.

It should be noted that although the Operational view was developed to support the demonstration of OHA in a top down design approach the model and the requirement set would have be developed concurrently.

The AFMS provides command and control and deployment of aerial firefighting assets within a larger system called the Bushfire Fighting Management System (BFMS). The AFMS consists of a command and control (C2) node and a network of airborne nodes known as Airborne Firefighting Control System(s) (AFCS). For the purposes of demonstration, the OHA demonstrated within this paper was confined to the AFCS with the AFMS treated as an external system.

The operational view the AFCS consists of a Performer called the Airborne Node that performs a top level OA named OA.5 Deliver Aerial Payload. This top level OA for the AFCS is further decomposed using an Enhanced Functional Flow Block Diagram (EFFBD) notation. The EFFBD for OA.5 Deliver Aerial Payload in provided Figure 3.

For the purposes of this analysis is it assumed that:

- The notification of fire incident events at the AFMS are received from the BFMS and are considered to be external to the AFCS.
- The derivation of valid environmental, geographic, high value asset and vulnerability information is external to both the AFMS and the AFCS
- Localised air traffic control is not required within an area of operation due to the low number of deployed aerial assets.

.

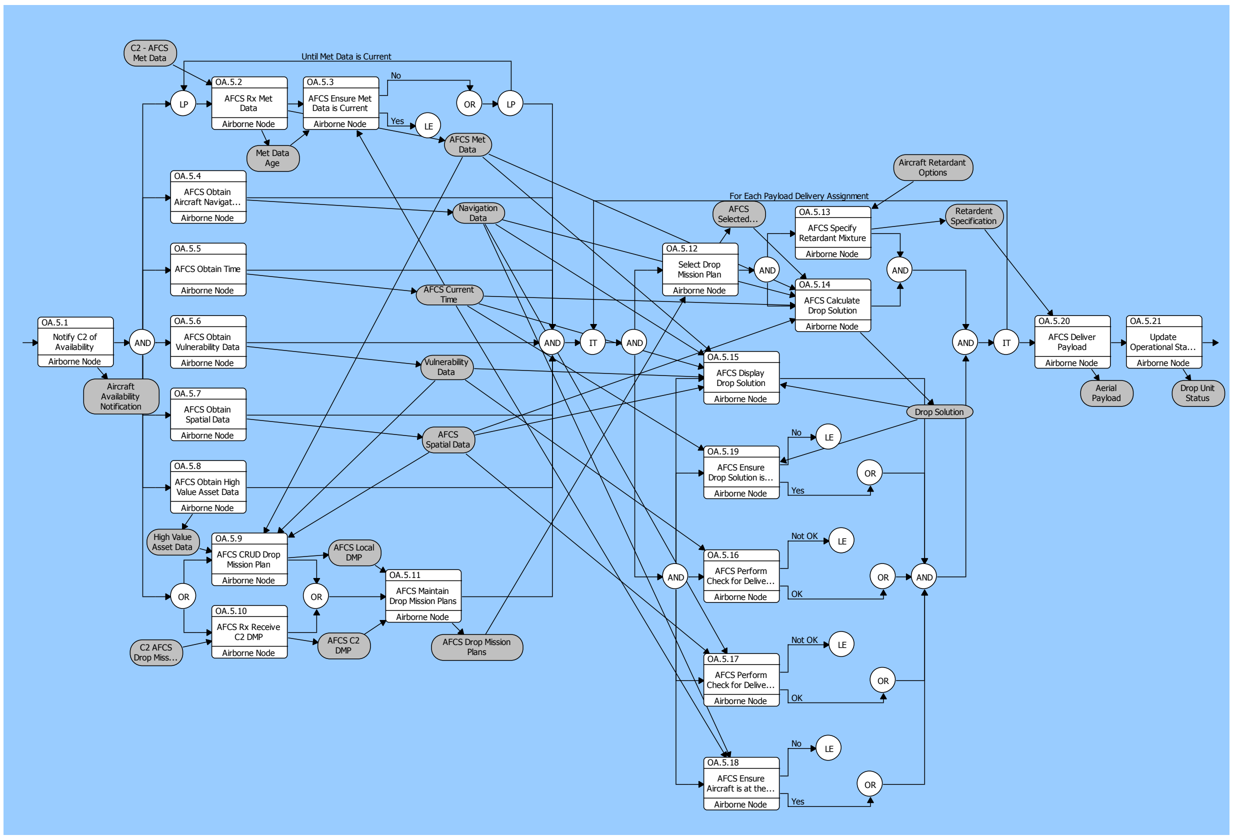

C2 - AFCS Met Data
Until Met Data is Current
OA.5.2 AFCS Rx Met Data Airborne Node
OA.5.3 AFCS Ensure Met Data is Current Airborne Node
No
Yes
LP
OR
LE
Met Data Age
AFCS Met Data
OA.5.4 AFCS Obtain Aircraft Navigat... Airborne Node
Navigation Data
OA.5.5 AFCS Obtain Time Airborne Node
AFCS Current Time
OA.5.1 Notify C2 of Availability Airborne Node
Aircraft Availability Notification
AND
OA.5.6 AFCS Obtain Vulnerability Data Airborne Node
Vulnerability Data
OA.5.7 AFCS Obtain Spatial Data Airborne Node
AFCS Spatial Data
OA.5.8 AFCS Obtain High Value Asset Data Airborne Node
High Value Asset Data
OA.5.9 AFCS CRUD Drop Mission Plan Airborne Node
AFCS Local DMP
OA.5.10 AFCS Rx Receive C2 DMP Airborne Node
C2 AFCS Drop Miss...
AFCS C2 DMP
OA.5.11 AFCS Maintain Drop Mission Plans Airborne Node
AFCS Drop Mission Plans
IT
Aircraft Retardant Options
For Each Payload Delivery Assignment
AFCS Selected...
OA.5.12 Select Drop Mission Plan Airborne Node
OA.5.13 AFCS Specify Retardant Mixture Airborne Node
Retardent Specification
OA.5.14 AFCS Calculate Drop Solution Airborne Node
OA.5.15 AFCS Display Drop Solution Airborne Node
Drop Solution
OA.5.19 AFCS Ensure Drop Solution is... Airborne Node
OA.5.16 AFCS Perform Check for Delive... Airborne Node
Not OK
OK
OA.5.17 AFCS Perform Check for Delive... Airborne Node
OA.5.18 AFCS Ensure Aircraft is at the... Airborne Node
OA.5.20 AFCS Deliver Payload Airborne Node
Aerial Payload
OA.5.21 Update Operational Sta... Airborne Node
Drop Unit Status

Figure 3 Deliver Aerial Firefighting Response EFFBD

6.1 Identifying the Safety Risk

The analysis of the OA decomposition, in Figure 3 seeks to identify the OAs where the failure mode of that OA can, possibly with the participation of one or more external co-effectors, directly result in an Accident. These failure modes of the OA are designated as Operational Hazards.

For each identified Operational Hazard a Hazard Sequence table is constructed to document how the failure modes of any other OA may contribute to the Operational Hazard. Were the failure mode for an OA can contribute to an Operational Hazard it is listed in the table.

While the analysis of Figure 3 identified several Operational Hazards for the AFCS, including Deliver Unsafe Payload, Failure to Deliver Payload and Define Unsafe Retardant Mixture, for the purposes of demonstration, this paper shall only consider a single Operational Hazard, namely OH.5.20 Delivery Unsafe Payload that is created by OA.5.20 Delivery Aerial Payload.

Table 1 provides the (abbreviated) Hazard Sequence Table for the Operational Hazard OH.5.20 Delivery Unsafe Payload identifying the failure mode of each OA in Figure 3 that creates a Contributing Factor that can participate in the Hazard Sequence for OH.5.20 Delivery Unsafe Payload.

Operational Activity	Contributing Factor
OA.5.2 AFCS Rx Met Data	CF.5.2 AFCS Fails to Receive Met Data
OA.5.3 AFCS Ensure Met Data is Current	CF.5.3 AFCS Fails to Ensure Met Data is Current
OA.5.4 AFCS Obtain Aircraft Navigation Data	CF.5.4 Obtain Inaccurate Aircraft Navigation Data
OA.5.5 AFCS Obtain Time	CF.5.5 Obtain Inaccurate Time
OA.5.6 AFCS Obtain Vulnerability Data	CF.5.6 Obtain Inaccurate Vulnerability Data
OA.5.7 AFCS Obtain Spatial Data	CF.5.7 AFCS Obtain Incorrect Spatial Profile
OA.5.8 AFCS Obtain High Value Asset Data	CF.5.8 Obtain Inaccurate High Value Asset Data
OA.5.9 AFCS CRUD Drop Mission Plan	CF.5.9 AFCS CRUD Unsafe Drop Mission Plan

Table 1 Hazard Sequence Table for OH.5.20 AFCS Deliver Unsafe Payload

Figure 4 provides a pictorial representation of the Hazard Sequence developed from Table 3 that shows the logical and temporal dependencies between Contributing Factors - shown in orange and the Operational Hazard - shown in red. Within the DR traceability is maintained between the Operational Hazards or Contributing Factors and the OA.

The existing DoDAF traceability between the Operational and System Views can then support an impact assessment of changes at the operational level on the safety of the implementation, or conversely, to assess the impact of changes in the implementation on the operational level safety.

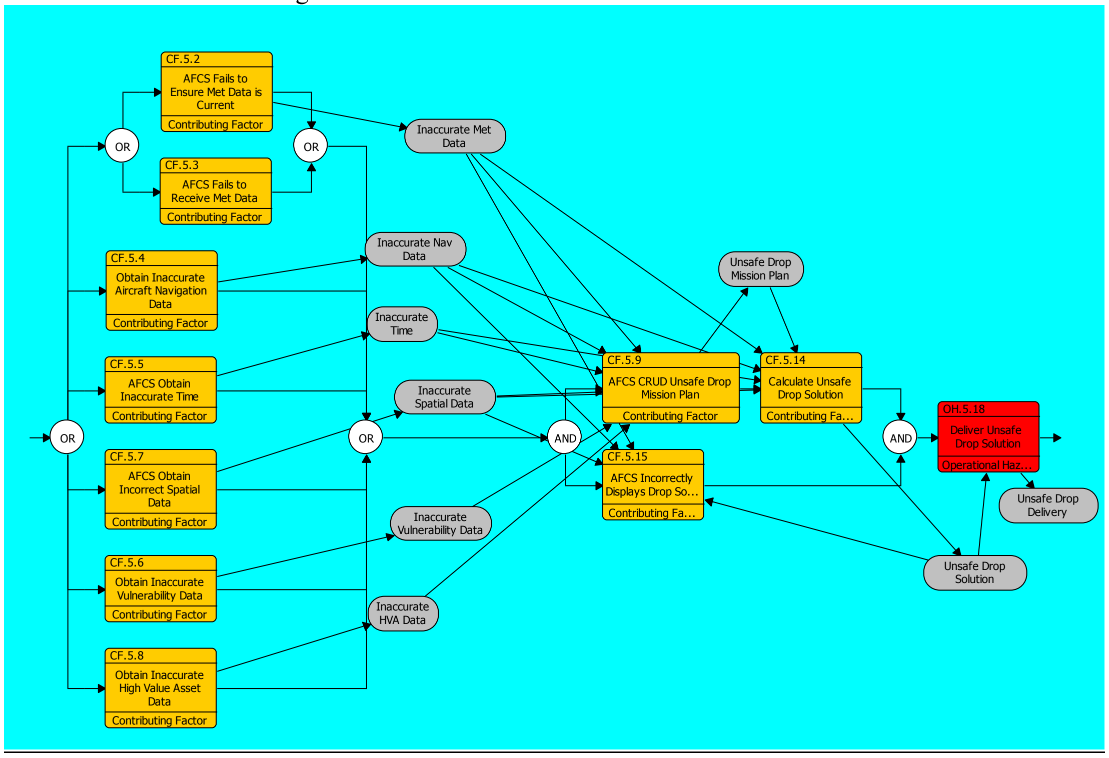


Figure 4 Hazard Sequence FFBD for Deliver Unsafe Payload

6.2 Identifying and Defining Operational Control Measures and Systems Safety Requirements

Analysis and management of the information and control dependencies within the Operational Views (that are also reflected in the hazard sequences) allows operational level Preventative Control options to be identified and incorporated into the operational concept. For example analysis of the hazard sequence for the AFCS in Figure 4 highlights the importance of accurate time, navigation, meteorological and spatial data to the determination of a safe drop solution and the safe delivery of an aerial payload.

Other risks that need to be incorporated into the safety analysis and the development of the SSR include.

- That the aircraft is not at the correct location for the drop
- That meteorological data currency issues are introduced by extended pauses within the drop release sequence.
- Failure to identify potential obstacles to the drop delivery.
- The existence of vulnerabilities that may be impacted by the delivery of a drop solution.
- Failure to identify of high value assets that require priority responses or special treatment.

Figure 5 extends Figure 4 by including a set of operational level Preventative Controls into the Hazard Sequence to address the risks described above.

Each Preventative Control then becomes the basis of a SSR as shown in Table 2.

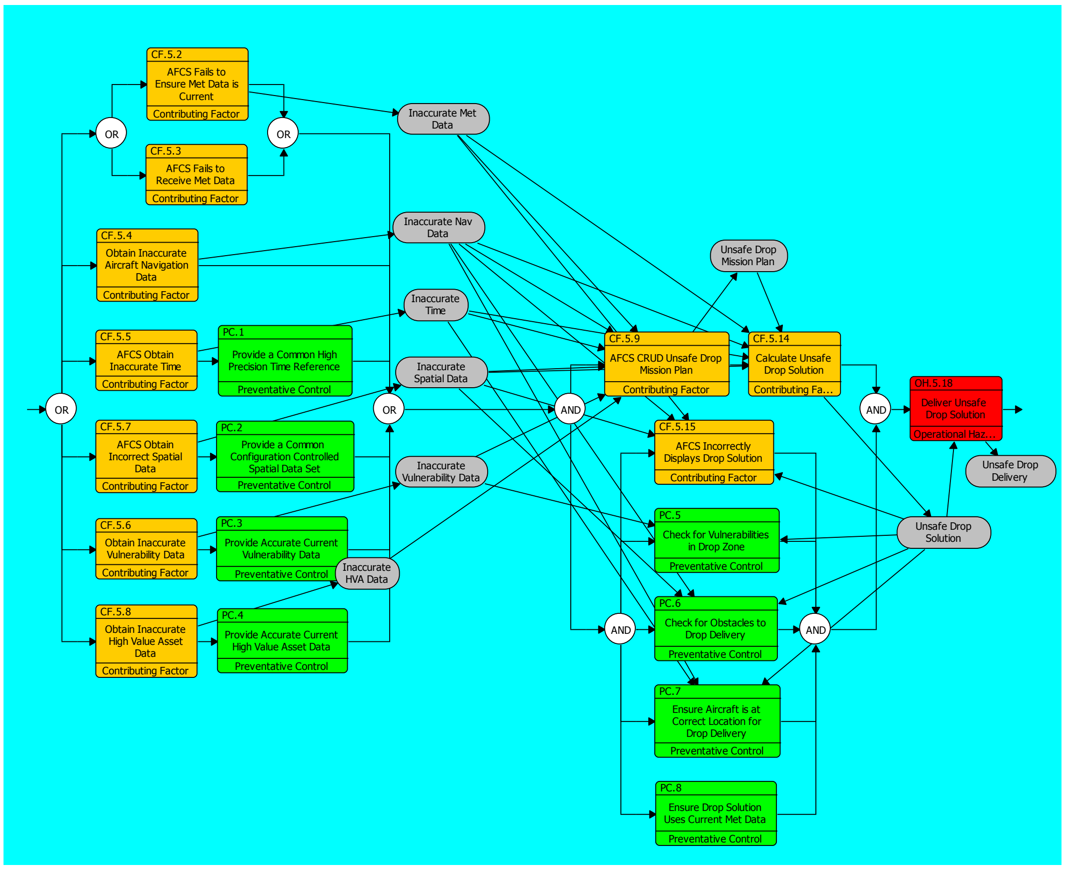

CF.5.2
AFCS Fails to Ensure Met Data is Current
Contributing Factor
CF.5.3
AFCS Fails to Receive Met Data
Contributing Factor
Inaccurate Met Data
Inaccurate Nav Data
CF.5.4
Obtain Inaccurate Aircraft Navigation Data
Contributing Factor
Inaccurate Time
CF.5.5
AFCS Obtain Inaccurate Time
Contributing Factor
PC.1
Provide a Common High Precision Time Reference
Preventative Control
Inaccurate Spatial Data
CF.5.7
AFCS Obtain Incorrect Spatial Data
Contributing Factor
PC.2
Provide a Common Configuration Controlled Spatial Data Set
Preventative Control
Inaccurate Vulnerability Data
CF.5.6
Obtain Inaccurate Vulnerability Data
Contributing Factor
PC.3
Provide Accurate Current Vulnerability Data
Preventative Control
Inaccurate HVA Data
CF.5.8
Obtain Inaccurate High Value Asset Data
Contributing Factor
PC.4
Provide Accurate Current High Value Asset Data
Preventative Control
OR
AND
Unsafe Drop Mission Plan
CF.5.9
AFCS CRUD Unsafe Drop Mission Plan
Contributing Factor
CF.5.14
Calculate Unsafe Drop Solution
Contributing Fa…
OH.5.18
Deliver Unsafe Drop Solution
Operational Haz…
Unsafe Drop Delivery
CF.5.15
AFCS Incorrectly Displays Drop Solution
Contributing Factor
Unsafe Drop Solution
PC.5
Check for Vulnerabilities in Drop Zone
Preventative Control
PC.6
Check for Obstacles to Drop Delivery
Preventative Control
PC.7
Ensure Aircraft is at Correct Location for Drop Delivery
Preventative Control
PC.8
Ensure Drop Solution Uses Current Met Data
Preventative Control

Figure 5 Hazard Sequence FFBD for Deliver Unsafe Payload showing Operational level Preventative controls

Preventative Control	System Safety Requirement
PC.1 Provide a Common High Precision Time Reference	A common high precision time reference shall be used by all AFCS functions
PC.2 Provide a Common Configuration Controlled Spatial Data Set	A common configuration controlled high integrity spatial data set shall be provided at all AFCS nodes
PC.3 Provide Accurate Current Vulnerability Data	Current accurate vulnerability data shall be provided to all AFCS nodes
PC.4Provide Accurate Current High Value Asset Data	Current accurate high value asset data shall be provided to all AFCS nodes
PC.5 Check for Vulnerabilities in Drop Zone	Operator Checks shall be performed to identify any vulnerability in the drop zone prior to the drop delivery.
PC.6 Check for Obstacles to Drop Delivery	Operator Checks shall be performed to identify any obstacles to the delivery of the drop solution
PC.7 Ensure Aircraft is at Correct Location for Drop Delivery	Drop solutions shall only be able to be executed when the aircraft is at the correct location
PC.8 Ensure Drop Solution Uses Current Met Data	Only a drop solution has been calculated with current meteorological data shall be able to be delivered.

Table 2 Preventative Controls and corresponding SSR

6.3 Accident Analysis

Accident	Hazard Sequence	External Co-effectors
Personnel Injury or Death from Drop Delivery	CRUD Unsafe Drop mission Plan	Personnel in Drop Zone

Table 3 Accident Sequence Table for Drop Delivery Causes Personal Injury or Death

Table 3 describes the Accident Sequence for Drop Delivery Causes Personal Injury or Death demonstrating how an Accident can result from a Hazard Sequence, with the contribution of one or more External Co-effectors. This relationship is graphically shown in Figure 6.

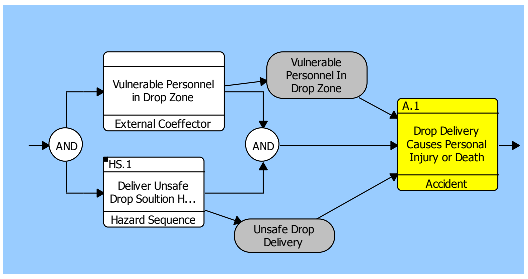

Vulnerable Personnel in Drop Zone
External Coeffector
Vulnerable Personnel In Drop Zone
AND
AND
HS.1
Deliver Unsafe Drop Soultion H...
Hazard Sequence
Unsafe Drop Delivery
A.1
Drop Delivery Causes Personal Injury or Death
Accident

Figure 6 EFFBD for Drop Delivery Causes Personal Injury or Death Accident Sequence

7 Summary

The OHA method within this paper includes a hazard analysis process that can be incorporated into the top down MBSE design processes that utilises the operational views of a capability to perform an implementation agnostic safety assessment that can drive the definition of the implementation agnostic SSR for the capability.

A Hazard analysis that employs commonly used architecture frameworks, such as DoDAF, within a commercially available MBSE DR allows the integration of the hazard analysis with the conceptual design activity and the management of the SSR within the mainstream MBSE reqiuirements management processes.

The OHA method enables analysis of each OA within a model of the operational view to identify the impact of the failure mode of each OA or to identify where each OA may provide a hazard reduction function within the overall capability operation.

At this stage the OHA method does not provide a complete hazard analysis as it is not currently integrated with the identification and management of implementation specific hazards such as power or component failure and their impact on system safety. While the outputs of the OHA process provide the context to evaluate the impact of system level hazards, such as those mentioned above, further investigation is required to formalise the traceability from the artefacts of OHA to those of the implementation level hazard analysis performed later in the design cycle.

This paper proposes that OHA and the use of an MBSE DR can provide a means to address many of the issues identified by Kelly and Allenby, such as the lack of an adequate system description to support hazard analysis at the beginning of the design cycle and the ability to integrate the safety engineering and SSR management within the mainstram systems engineering processes.

8 Conclusions

Integrating the safety assessment into the initial MBSE capability concept development process creates a 'living' top down framework to identify and mitigate risk early in the design cycle. An MBSE DR can also support the capture and reuse of information throughout the lifecycle and its subsequent reuse within future engineering iterations.

The operational views of the capability architecture enable a top down safety evaluation to be performed on a design representation that can be shared by both the mainstream systems design and safety engineering, This sharing of data can enhance communication and coordination within the design team while minimising the likelihood of introducing errors that can occur when the work of each specialty area utilises separate, and therefore possibly inconsistent, systems representations.

9 Future Work

Further research is required to develop traceability from the Operational Hazards and Contributing Factors identified in the OHA to those hazards that arise from an aspect of the implementation.

A potential method of developing a qualatative risk rating of each Operation Hazard and Contributing Factor based on the information dependencies and functionality under consideration could also be devloped.

Possible further research could also be performed to develop a verb - noun convention for the naming of OAs with a corresponding set of failure mode templates defined for each verb term. These templates could be used to assist in the identification of the safety risk associated with the OA.

10 Acknowledgments

This work was undertaken as part of the author's PhD project at The University of Queensland.

The author would like to acknowledge the guidance and support provided by Dr Clive Boughton and Professor Peter Lindsay both in their roles as my PhD supervisors and also in the development of this paper.

11 References

1. Defence Capability Development Manual; Australian Government Department of Defence, (2006) http://www.defence.gov.au/publications/DefenceCapabilityDevelopmentHandbook2012.pdf Accessed: 25 May 2014
2. National Defence Industrial Association (NDIA), Final Report of the Model Based Engineering (MBE) Subcommittee (2011). http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Committees/M_S%20Committee/Reports/MBE_Final_Report_Document_%282011-04-22%29_Marked_Final_Draft.pdf
3. Kelly, T. Allenby, K.(2001) Deriving Safety Requirements Using Scenarios Proceedings Fifth IEEE International Symposium on Requirements Engineering, 2001.
4. Stringfellow Herring, M. et al. Safety-Driven Model-Based System Engineering Methodology Part I: (2007) http://sunnyday.mit.edu/JPL-Part-1.pdf
5. Leveson, N. Intent Specifications; An Approach to Building Human-Centered Specifications. IEEE Transactions on Software Engineering, VOL. 26, NO. 1, (2000).
6. Leveson, N. A New Accident model for Engineering Safer Systems, Safety Science,

Volume 42, Issue 4, April 2004, Pages 237-270 http://www.sciencedirect.com/science/article/pii/S092575350300047X

7. Lindsay, P. A., et al. (2012). Model-based safety risk assessment using Behavior Trees. Asia Pacific Conference on Systems Engineering (APCOSE)/Australian Systems Engineering, Test & Evaluation (SETE) 2012 combined conference, Systems Engineering Society of Australia. http://www.itee.uq.edu.au/sse/afms

8. United States Department of Defence Architecture Framework (DoDAF) dodcio.defense.gov/Portals/0/.../DODAF/DoDAF_v2-02_web.pdf

A Process for Integrating Alarm Systems for Operation, Safety and Security in Process Plants

Kourosh Parsa and Peter A. Lindsay
School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, Queensland 4072, Australia
kourosh.parsa@outlook.com, p.lindsay@uq.edu.au

Abstract:

Alarm systems play a vital role in a process plant, helping the operator to keep process variables within normal operating range and maintaining safety of the plant. Often multiple alarm systems are in operation concurrently, because they concern different aspects of the plant (process control, safety or security), or because they have been installed as part of the equipment provided by suppliers. For ease of operation, and efficiency and effectiveness of response, it is desirable to integrate the different alarm systems into a single Alarm Management System (AMS), rationalized to prioritize how alarms are presented to operators and to improve diagnosis and treatment.

This paper studies the functionality of alarm management systems in process plants and details alarm system requirements from the various process control, functional safety and security standards that apply. In particular, it discusses the interaction between ISA-18, IEC-61508, IEC-61511, and ISA-99. Functional security breaches can undermine functional safety measures and are almost impossible to prevent, so an AMS must be designed to detect anomalies and enable forced shutdown of systems. Standards give very little guidance on how to react once functional security breaches are detected. This paper proposes an approach integrating functional security measures into the functional safety lifecycle.

The approach is illustrated on two case studies: the BP Texas explosion in 2005 and the Stuxnet malware incident in 2010. [1]

Keywords: Alarm Management System, Functional safety, Functional security, Malware, Cyber security.

1 Introduction

The purpose of an integrated alarm system in process plants such as refineries, petrochemical and power plants is to improve the functional safety and security of the plant's operation. Typically a plant will have many different alarm systems because they concern different aspects of the plant or because they have been installed as part of equipment provided by different suppliers. This paper discusses the process of integrating them into a single Alarm Management System (AMS). Using a lifecycle approach, this paper critically overviews AMS requirements to ensure that functional safety and security are addressed in an integrated manner.

A range of different standards apply to AMSs. IEC-61508:2010 is the generic functional-safety standard for AMS hardware and software; IEC-61511: 2003 is its instantiation for the process industries. ANSI/ISA-18:2009 is the standard for management of alarm systems. ANSI/ISA-99:2007 is the standard for functional security.

The paper is structured as follows: Firstly, the alarm management lifecycle stages are explained and AMS components are introduced. Then important alarm characteristics are identified, such as their associated time-line, technology nature, and place in the safety protection layer hierarchy. The next step is to extract AMS requirements from functional safety and security standards and best practice, and integrate them into a plant operational lifecycle model.

The paper concludes by applying the approach after-the-fact to two case studies, to illustrate how the incidents might have been prevented.

Table 1. Acronyms

AMS	Alarm Management System
BPCS	Basic Plant Control System
FGS	Fire & Gas System
HMI	Human Machine Interface
IPL	Independent Protection Layer
SIF	Safety Instrumented Function
SIL	Safety Integrity Level
SIS	Safety Instrumented System

2 Alarm Management System Functionality

To design an effective alarm management system it is necessary to determine the different functionalities in operation, safety and security. In fact, it is critical to distinguish between normal and abnormal situations in each of these areas.

Nowadays, alarm management systems have greater capacity to improve product quality, reduce unplanned shutdowns, and prevent environmental damage, injuries and fatalities (EEMUA, 2007). It depends, however, on how well alarms have been designed and how effectively the alarm system has been managed. As a matter of fact, to ensure the effectiveness of AMS, each stage of the alarm management system lifecycle must be studied.

The ANSI/ISA-18 standard, "Management of alarm system for process industries", specifies 10 stages for the alarm management lifecycle (Fig. 1):

1. Alarm Philosophy: Develop a roadmap about required alarms and their characteristics, such as alarm set points and limits.
2. Identification: Identify all alarms that will be installed at the plant.
3. Rationalization: Justify the need for each alarm, and prioritize and group alarms to increase alarm system performance and efficiency.
4. Detailed Design: Complete the AMS design, including its Human Machine Interface (HMI).
5. Implementation: The alarm system is installed and commissioned.
6. Operation: The alarm system is in operation and operators control the alarms.
7. Maintenance: One or more alarms are out of service or under periodic test.
8. Monitoring and assessment: Review alarm system performance and capabilities based on the alarm philosophy.
9. Management of change: Verify and validate any plant or alarm modification.
10. Audit: Undertake periodic audit reviews to ensure alarm system effectiveness.

Although the alarm management lifecycle provides guidance for developing an effective alarm system, it is also crucial to study alarm system components. This helps to verify if the system is integrated and functionally safe and secure.

3 Alarm Management System components

Process variables are measured by sensors and transferred through communication channels to control systems and the associated HMI. The operators make decisions according to the data available through HMI and the plant operation manual. The decision is applied to the process operation through the related process control system and final control elements (Timms 2009).

Alarm management components include (Fig. 2):

- Instruments such as sensors and final elements
- Systems, including the Basic Plant Control System (BPCS), the Safety Instrumented System (SIS), and the Fire and Gas System (FGS)
- Network elements, such as switches and communication media
- Logic, including software programs
- The Human Machine Interface

To design an effective AMS, these components must work as a fully integrated system and meet the functional safety and security requirements.

Under normal all-is-well operation, control loops in the BPCS control process variables automatically, based on pre-defined logic. However, because of unpredicted and complex combinations of events, the AMS must include humans in alarm loops. The operator needs to be given enough information to be able to diagnose the reason for the alarm, take the appropriate action based on the plant operation manual, and then check whether the action has been effective. This increases the functional complexity of the total system.

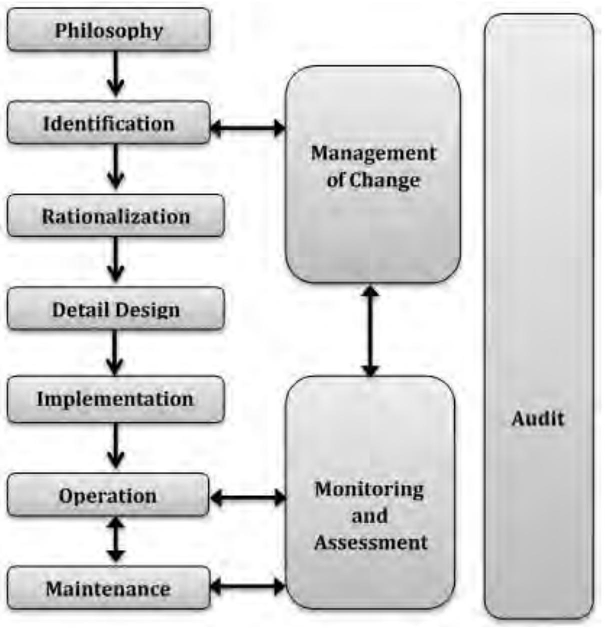


Figure 1: Alarm Management Lifecycle (ANSI/ISA-18, 2009)

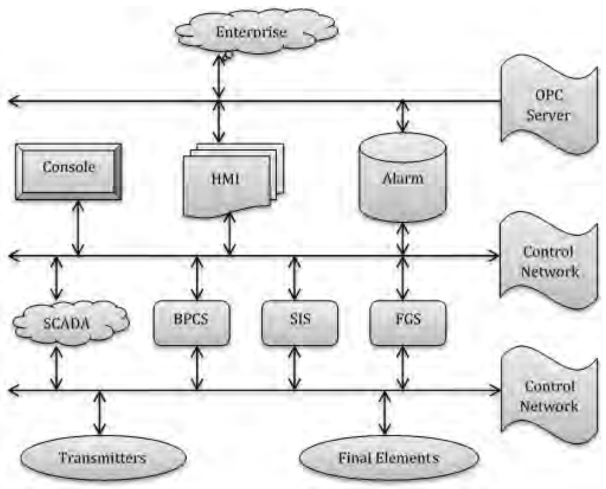


Figure 2: AMS Dataflow

4 Alarm Identification

The first step in our proposed AMS development approach is to define alarm states, the target area of plant operation, the normal approach area, upset area, alarm area and shutdown area (Fig. 3). The marginal part of each area is the critical part of alarm definition, because these areas are important for alarm issues such as chattering or standing alarms. The criteria of this step are normally defined by the plant alarm philosophy or related standards.

Alarms are identified by a variety of engineering practices or regulatory requirements (ANSI/ISA-18 2009). Plant development techniques that identify alarms include:

- Allocation of safety layers
- Preliminary Hazard analysis (PHA)
- Hazard & Operability (HAZOP) analysis
- Layer of protection analysis
- Environmental permits

- Failure Modes And Effects Analysis
- Piping and Instrumentation Diagram (P&ID) reviews
- Operating procedure reviews
- Package manufacturer recommendations

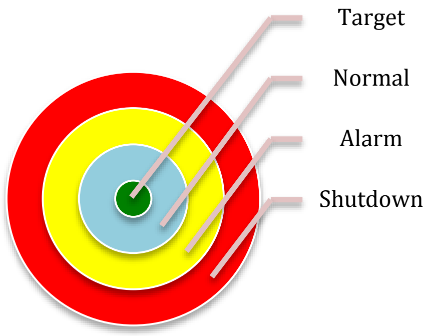


Figure 3: Process Variable State (ANSI/ISA-18, 2009)

Alarm and Trip development is a step-by-step process which is done based on the process requirements or hazardous scenarios in HAZOP. However, they are generally not sufficient to ensure an effective integrated AMS. For example, they do not consider alarm time-line management and alarm response.

5. Alarm Time-line and response model

An alarm loop has different components with different timing characteristics. Calculating maximum total response time provides an estimate of how much time is required for an alarm loop, in comparison to the time required for process operation for that particular loop. ANSI/ISA-18 develops a feedback model for process operator interaction. In this model, three main parts are involved, the required time to detect the alarm state, operator diagnosis of the alarm, and responding to the alarm.

Standard ANSI/ISA-18 introduces a response model to demonstrate how the process variable moves towards the alarm state with and without operators' responses (Fig. 4). Firstly, if the process variable is in the normal area, then it is in the un-acknowledged area which means the alarm has not been activated or it is being processed by the AMS. It is then the operator's turn to make a timely decision. Due to the process dead-time, the process variable takes some time to respond to the operator's action. After this stage, the process variable returns to its normal condition. This model shows that if the operator does not take the required action, the process variables will move towards the trip area.

6. Plant layers of protection

IEC-61511 defines a plant layer of protection intended to increase the plant's level of safety (Fig. 5). The layer of protection consists of two categories, prevention and mitigation. Alarm performs in the first two layers of the prevention category (BPCS & SIS) (Stauffer 2012).

Alarms, as a part of the Independent Protection Layer (IPL), must have the following properties:

Specificity: Each IPL is designed solely to prevent or mitigate the consequences of potentially hazardous events.
Independence: Each IPL is independent of the other protection layers associated with the identified danger.
Dependability: The IPL must be accountable for what it is designed to do. Both random and systematic failure modes are addressed in the design.
Auditability: The IPL must be designed to facilitate regular validation of the protective functions.

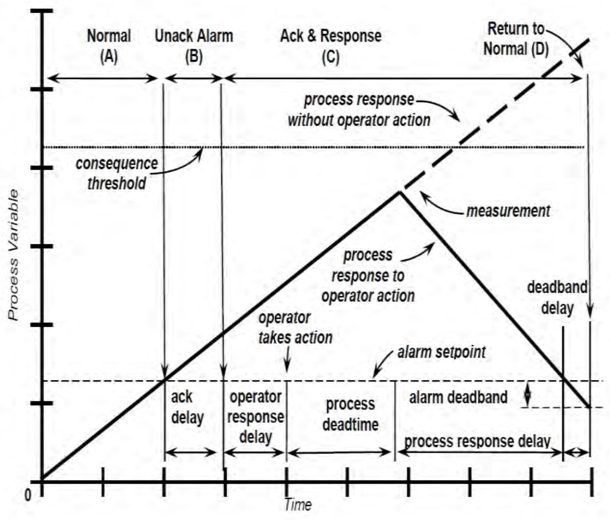


Figure 4: Alarm Time Line (ANSI/ISA-18, 2009)

Proof testing and maintenance of the safety system is necessary. Only those protection layers that meet the tests of availability, specificity, independence, dependability, and auditability are classified as independent protection layers (IEC-61511, 2003).

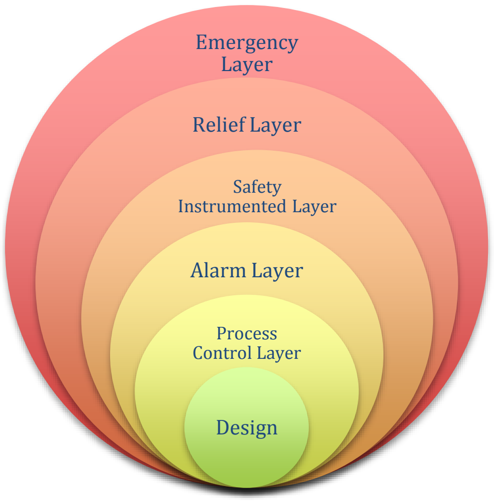


Figure 5: Layers of Protection

7. Safety Instrumented Functions

Alarms are part of the Safety Instrumented Functions (SIFs) to reduce process operation risk to tolerable risk (ANSI/ISA-18, 2009). For this kind of alarm, the required risk reduction factor (RRF) must be calculated. Also, the required Safety Integrity Level (SIL) must be assigned to alarm loops. The question here is what the risk reduction factor for the operator would be. EEMUA-191 recommends RRF=0.1 for experienced operators without stress. Finally, to complete SIL calculation for the alarm loops, hardware and software requirements will be studied (IEC-61508, 2010).

8. Alarm Loop Hardware and Software

To comply with IEC-61508 all the components of the alarm loop must be separated from other control or safety loops.

To achieve the required Safety Integrity Level (SIL), sometimes an alarm loop must have redundant sensors or a voting system. However, it is necessary for operators to deal with only one alarm and not redundant alarms as any redundant or parallel alarms can confuse the operators. So redundant alarms must be processed before visualizing in the HMI.

9. Human Machine Interface (HMI)

The HMI is a critical part of the AMS, as it notifies operators about abnormal situations. To improve the functionality of HMI, it is recommended that the pop-up text messages must be clear and concise. Also, a process loop must be located entirely in one HMI. In addition, figures and symbols must be understandable and active alarms must be clearly distinguishable from non-active alarms.

Finally, issues that occur during the operation can cause risks for AMS functional safety. Alarm issues such as alarm flood and alarm chattering affect the operator's ability to respond in a timely and appropriate manner. The aim of alarm rationalization and prioritization is to reduce the likelihood of such issues occurring.

10. AMS and functional security

One question is how to ensure that an alarm management system is functionally secure. The next question is what actions must be taken when the alarm management system is affected.

Firstly, the data flow architecture of the alarm management system must be reviewed. The components which are involved in the data flow transmission, such as network mediums (for example wireless systems), network switches, data servers, and alarm record systems must be segregated from upper layers such as the business enterprise and process plant management layers. In addition, a firewall must be placed between layers to control the data flow. Furthermore, the alarm network must be designed in divided zones to reduce the probability of affecting the alarm network system. (ANSI/ISA-99, 2007).

A network communication protocol has a considerable impact on network intrusions. For example, when the communication protocol is in the form of Ethernet, the risk of network intrusion will increase. Also, where the data is transferring through an OPC server (OLE for process control) for recording alarms, or visualizing them in multi-functional applications or in HMI, potentially the risk of hacker and worm intrusions will be very high. Indeed, it can negatively impact the AMS functionality.

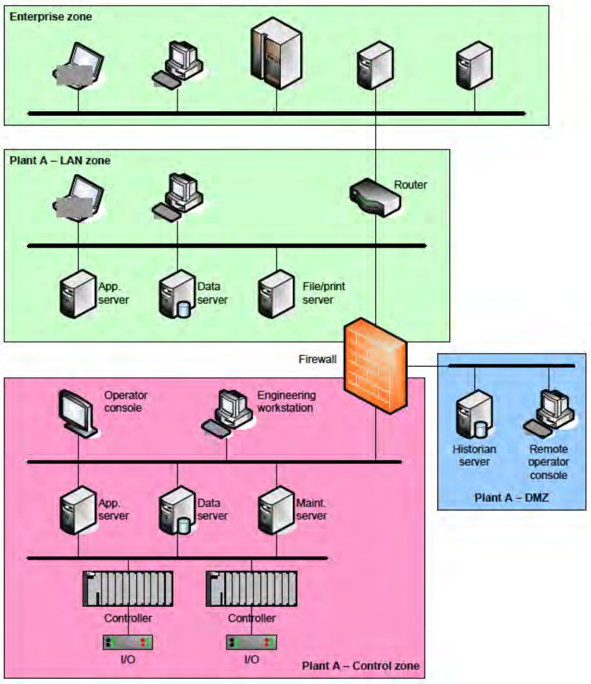


Figure 6: Cyber security zones

In addition, Industrial Cyber Security (ICS) divides network threats into two categories, active threats and passive threats. Active threats include network intrusions that can be performed by hackers and viruses. Passive threats include malware, such as Trojans and worms, which operate inside the network and affect the data (ANSI/ISA-99, 2007).

ISA-99 recommends a few steps to increase the security of industrial automation and control networks. These steps involve the design of the network architecture, as mentioned before, and firewall configuration between layers of dataflow (Fig. 6). It also defines the access control level in order to authorize and authenticate users and interconnects different process control sub-networks. Another step is to increase access control in wireless communication and to use redundant channels for alarms in SCADA systems in particular. Controlling and updating anti-virus software is another step recommended by ISA-99. Any network, including wireless communication, must be segregated from other sub-networks as they are at high risk of intrusion (Vinar, M.I. et al., 2006).

10. An integrated approach

Table 2 is developed to address the activities to meet functional safety and security requirements based on the ANSI/ISA-18 alarm management lifecycle. These activities are to increase safety and security of the Alarm Management System.

AMS data communication through wireless and the Internet must have sufficient encryption and protection code against intrusion or data defection.

The compatibility of all communication protocols must be checked for reliability and access for all related hardware, to ensure the protocol communicates functionally, safely and securely.

Table 2: An integrated AMS development process

Functional Operation	Functional Safety	Functional Security
	Alarm Philosophy	
Plant alarm Guide	Process control and HSE requirement	Define required Alarm Availability, Integrity and confidentiality. Define required network architecture
	Identification	
Define possible alarms	PHA, HAZOP and SIL identification	Cyber security risk assessment, Threat model, Define AMS Network architecture, Identify security alarms, Identify required alarms for mediums.
	Rationalization	
Group and prioritize alarms	Prioritize alarms, Diversity of alarms types in SIFs	Prioritize Security Alarms
	Detail Design	
Develop Alarms through engineering steps	Alarm Timing calculation, Alarm Functionality review in Plant Layer of protection. Alarm review	Design Network zone, Required Firewall. Develop Authentication and Authorization system, Design Alarm backup system
	Implementation	
Implement alarms in IPCS and design HMI	Implement Alarm in the plant-integrated system, Analyze redundant alarms or voting alarms before visualization. AMS PSSR	Cyber Security FAT, Cyber Security SAT and Startup. Block all un-used ports. Implement Access control system.
	Operation	
Alarm respond by operator	Proof test of BPCS and SIS to maintain Functional safety requirements	Control access, Update anti viruses, Service Packs and Firmware
	Maintenance	
Review Alarm characteristics	Alarm Loop check	Check AMS Data availability and Integration
	Monitoring and assessment	
Evaluate alarm issues	Update Alarm database, Record Alarm issues	Update Vulnerability assessment
	Management Of Change	
Alarm modification review	Update SIL calculation and SIF design	Update Threat model for changes and Verify AMS dataflow security
	Audit	
Update Alarm database	Proof test	Check the access control, Vulnerability records

All above-mentioned steps are taken to mitigate the risk of security threats. However, if the alarm management system is affected, there are two important issues. One is how to recognize that the system is affected and the second is how to react.

By considering the important role of the alarm management system in a process plant, if the alarms are affected, alarm verification becomes difficult. Also, if operators react to the affected alarms, possibly this response could cause an incident or lead to product loss, so it is critical to realize the system has been affected. After it is confirmed that the alarm management system is affected, any data transmission to the affected zone must be disconnected. The process unit in that area must be shut down following plant emergency shutdown procedures.

12. Case studies

12.1 Functional Safety: BP Texas Refinery Explosion

AMS, as a control loop in a process plant, plays a critical role in the safe control of the plant. Investigations of incidents, which were caused either because of inappropriate design of AMS or due to faulty operation, demonstrate the importance of functional safety in AMS. One of these incidents was an explosion in BP's Texas refinery in the Isomerization unit in 2005. This incident resulted in 15 deaths, 180 people injured and more than $1.5 billion damage. One of the reasons for the explosion was an AMS malfunction (Mogford 2005).

Following our proposed approach to design to improve functional safety of AMS, the solutions below could have been considered to prevent this incident:

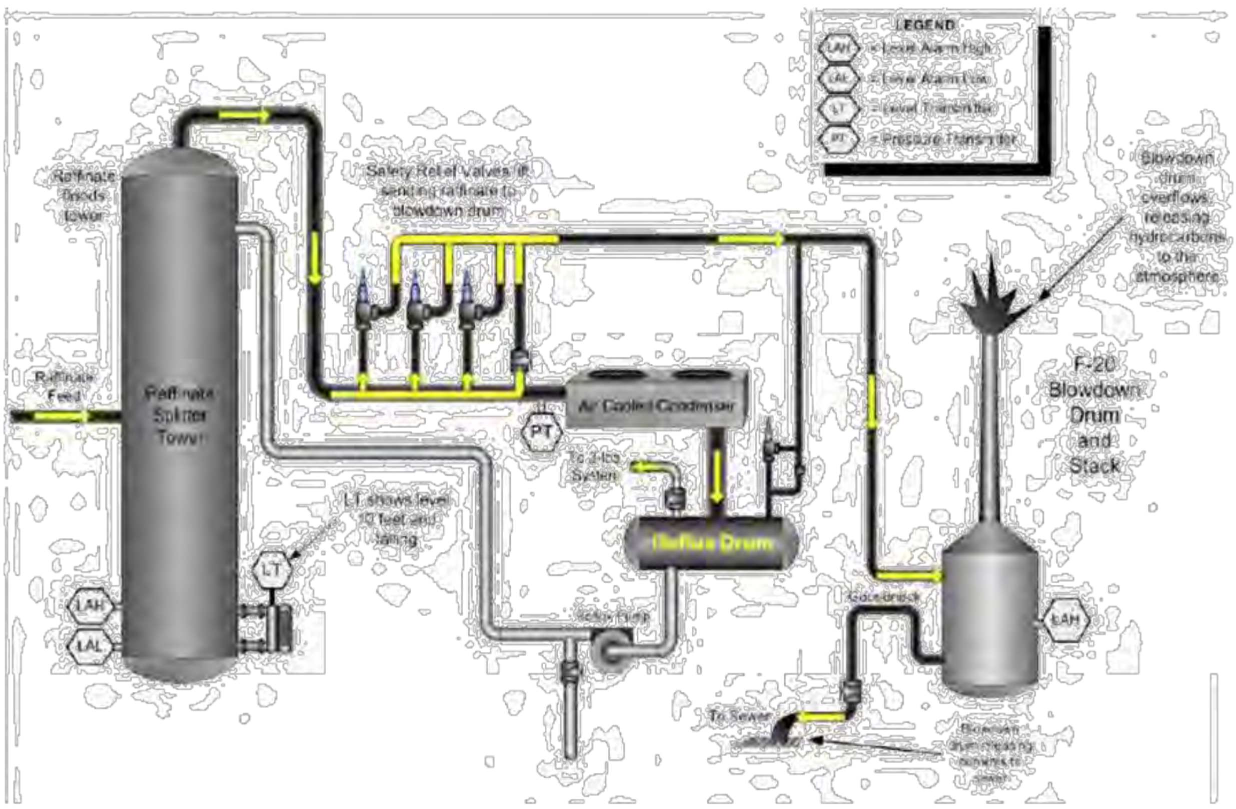


Figure 7: BP Explosion Process Flow (Mogford 2005)

- The main instruments which provide alarm signals, in particular, level transmitters on the Splitter and Drum, must be redundant to ensure that in the case of the failure of one instrument, the alarm signal will not fail so the operator will not lose the ability to make a proper decision.
- The use of different types of transmitters could help to reduce dependency on the process variable characteristic. In this incident, using a differential pressure transmitter on the splitter was inappropriate when the process liquid had variable viscosity. The reason is that the level transmitter had to recalibrate based on variable viscosities.
- The types of ESD level transmitters must be changed from contacting types, which depend on the installation method, to non-contacting types, to reduce dependency on the installation and to prevent failure, especially during plant start-up.

12.2 Functional Security: Stuxnet

The security of AMS, as an information management and control system, is critical to its operation. If an operator receives affected or delayed data, AMS will lose its effectiveness. This is because AMS's proper functioning depends on accurate and timely data. In addition, the operator will not be able to make the required decision, as the data are not reliable anymore, leading to the system losing its reliability. In process industries, modern control systems are implemented based on data networks. Therefore, data security is critical in order to prevent or reduce the probability of the penetration of malware and viruses into the system.

Industrial malware and viruses are divided into two categories. One includes those viruses in common with PC networks that affect the network communication or software performance. The second category includes the malware or viruses exclusive to industrial environments. This category affects system reliability through data destruction. In industrial networks the upper layers, such as the management layer, database and Internet connection, are the target point for the first category. Most of the malware and viruses from this category will be detected by current up-to-date anti-virus software, available in the market. The second category of malware and viruses are created specifically for industrial automation networks and mainly target control networks. This is the reason the detection of these kinds of malware and viruses is much more difficult, and they are usually detected after the system is affected. Therefore, this category is considered a big threat to the reliability of the system (Milevski 2011).

A recent example of the penetration of advanced malware to control a network, "Stuxnet", is discussed here in order to apply the lessons learnt from this case to increase the functional security of AMS. Stuxnet was recognized as industrial malware in Belarus in 2010. (Chen 2011) It is believed that it was spreading a few months before being detected. Stuxnet differs from previous malware in a number of characteristics. One was that Stuxnet was created exclusively to affect industrial automation software. The second characteristic was that this malware used two stolen security certificates from JMicron for its legitimacy. Stuxnet could enter the network through USB flashes, the Internet and SCADA and then spread into the network through network servers. Stuxnet attacked Siemens' control logic programs in industrial automation systems. In addition, it penetrated into the Siemens' programming software (PCS7) and HMI design programming (WinCC). (Chen 2011) Stuxnet used DLL (Data Link Library) files, which are used to introduce hardware to software, to get through to the control system network and affect the control logic, which caused operators to receive incorrect data. In addition, as a result of the affected control system, the motors' speed control system did not operate properly and consequently, speed control alarms did not activate correctly (Piggin 2011).

AMS must be protected not only against the penetration of malware and viruses, but the affected AMS must also be diagnosed as quickly as possible. Considering the involvement of AMS in different layers of operation, control and protection, achieving this objective becomes complicated. The recommendations provided by ISA-99 regarding preventing cyber security threats, such as network segmentation, using gateways between different zones and using anti-virus software for upper layers, are necessary but not sufficient. In order to increase the security of AMS, the affected system must be diagnosed and its safe state status must be defined after diagnosing the affected system.

One solution could be undertaking data traffic control in the network in order to detect any possible malware or virus in the network. Another solution could be to consider a data timing system when transferring data in the AMS network to ensure that operator receives data with no delay, then related command transfer to final element without delay. A third solution could be data coding to ensure the security of data transfer. Designing servers in parallel to decrease the malwares' spreading speed could be a fourth option.

However, considering all these solutions, if the developed malware or virus is compatible with the plant control system environment, as it happened in the case of Stuxnet, there is still a possibility of AMS being affected. This confirms that a mechanism must be devised to detect the affected AMS. One solution could be controlling the abnormal behaviour of alarm activation comparing to related process value, for example check the local indicator of the speeds and compare them with alarm level. Another solution could be controlling the behavioural pattern of alarm activation; if it is not compatible with process patterns or equipment characteristics then it could be a sign of the existence of a threat.

13. Conclusion

Plant productivity, safety and security all need to be taken into account when designing an integrated Alarm Management Systems. This paper has proposed an integrated approach to development of an AMS synthesising the requirements from the main process industry functional safety and functional security standards.

As recent events have shown, cyber security threats have significant potential to undermine safety and reliability of plant operation. Since it is difficult to protect against all security threats, it is necessary to also consider how to detect security breaches and react to them. For AMS as a control loop it is necessary to design and implement it to be functionally safe and as a data management system, it is necessary to keep it functionally secure. This is critical, since any cyber security threat not only reduces AMS efficiency but also reduces its reliability.

14. References

ANSI/ISA-18:2009, Management of alarm systems for the process industries, International Society of Automation.

ANSI/ISA-99:2007, Security for industrial automation and control systems, International Society of Automation.

Chen, T.M. & Abu-Nimeh, S., 2011, "Lessons from Stuxnet", Computer, vol. 21, no. 4, pp. 91-93.

EEUMA 2007, Alarm systems: A guide to design, management and procurement, Engineering Equipment and Materials Users Association, London.

IEC-61508:2010, Functional safety of electrical/ electronic/ programmable electronic safety-related systems, International Electrotechnical Commission.

IEC-61511:2003, Functional safety / safety instrumented systems for the process industry sector, International Electrotechnical Commission.

Igure, VM et al 2006, "Security issues in SCADA networks", Computers & Security 25, pp. 498 –506.

Milevski, L. 2011, "Stuxnet and strategy: a special operation in cyberspace?", Joint Force Quarterly 63, pp. 64.

Mogford, J. 2005, "Fatal Accident Investigation Report, Isomerization Unit Explosion" Final Report, Texas City, Texas, USA.

Piggin, R, 2011, Control network security lessons from Stuxnet, CFE Media, Barrington.

Stauffer, T, 2012, "Implementing an effective alarm management program", CEP Magazine, pp. 19-27.

Timms, C, 2009, "Hazards equal trips or alarms or both", Process Safety and Environmental Protection, vol. 87, no. 1, pp. 3-13.

Author Index

Recent Volumes in the CRPIT Series