Engineering Safety-Critical Complex Systems

Robert Alexander, Ruth Alexander-Bown, and Tim Kelly

Department of Computer Science
University of York, York, YO10 5DD, UK
{robert.alexander, tim.kelly}@cs.york.ac.uk

Royal United Hospital, Bath
ruth@alexander-bown.com

Abstract. Some of the complex systems with which the CoSMoS project is concerned are safety-critical, and if such systems are ever to be built and operated then they will need to be certified safe to operate. By looking at how conventional safety-critical systems are developed, we can find basic principles for safety-critical complex systems – this may be harder or easier than non-safety-specialists expect. In this paper, we outline current safety engineering methods and illustrate them using an artificial platelet case study. We also summarise our previous work on using simulation in safety engineering, and make some observations about applying simulation to very small systems.

1 Introduction

The CoSMoS project [1] is concerned with the engineering of complex systems. Some proposed complex systems are safety-critical – they have the potential to cause human injury or loss of life. Engineers developing safety-critical systems must meet a range of legal requirements, and to achieve this they must use considerable safety-specific technical knowledge. To outsiders (used to the engineering, particularly software engineering, of non-safety-critical systems) the methods used in safety-critical systems engineering may seem arcane, and in places even primitive or backwards. Nevertheless, failing to follow these methods may lead to the development of systems that can never be used because of safety concerns.

Sections 2 and 3 of this paper provide a primer in current safety-critical systems thinking for complex system researchers and developers. This is not a complete solution to safety concerns, or a cookbook approach to safety engineering. It does, however, provide the non-safety-expert with a starting point for appreciating the specific challenges that safety-critical complex systems pose. Equally, it should provide some insight into what is *not* needed under current safety regimes, and thereby open up avenues that may have seemed impractical before.

In Section 4, we have taken the Artificial Platelet (AP) system used as a case study by the TUNA project [2] and performed an initial hazard identification on the concept. We then use this to illustrate a number of the safety engineering issues raised in the previous section, and to highlight some of the problems that complex systems present.

To supplement the above, and with particular relevance to the simulation aspects of the CoSMoS project, Section 5 reviews previous work by some of the authors on simulation-based analysis of complex Systems of Systems (SoS). This illustrates how simulation techniques can move engineered systems from the unmanageable to the potentially tractable. Some of the extant challenges faced by this work will be equally (or more) salient to the CoSMoS project.

Finally, Section 6 discusses some of the unique aspects of engineering and simulation for very small artefacts.

The emphasis throughout is on the practical implications for complex system engineering (in terms of practical needs) and research (in terms of research objectives).

2 The Need for Safety Engineering

Safety engineering is relevant to a great many endeavours; specifically, any situation where human injury or death is possible. In the UK, the legal requirement to perform safety activities stems from the Health and Safety Executive and the 1974 Health and Safety at Work act [3]. For example, the HSE imposes a duty of care on employers with respect to the safety of their staff.

Certain industries, such as air transport and nuclear energy, are explicitly *regulated*, and specific legal duties are imposed on manufacturers and operators. These go beyond the common requirements imposed by the HSE; for the most part, the regulations are concerned with preventing major accidents. A recent overview of the relevant safety regulations for a number of different industries can be found in Jones [4]. A common theme is that before an installation or product can be operated or sold, it must be *certified* – an independent safety assessor must agree that it appears to be adequately safe.

Safety in regulated industries is governed by *safety standards* – documents that lay down what procedures must be followed in order to claim adequate safety (and thereby achieve certification). These standards are (generally) industry specific. For example, DO-178B [5] covers the safety of aircraft software, while Def Stan 00-56 [6] sets safety requirements for all UK military equipment.

Traditionally, most standards were *prescriptive*; they laid down a set of procedures to follow and processes to perform. If a product developer followed the process, they could claim an adequate safety effort. For example, a software-related standard might mandate code reviews, MC/DC test coverage, and the use of a programming language subset designed to eliminate common coding errors. Examples of prescriptive, process-based standards are DO-178B and the obsolete Def Stan 00-56 Issue 2 [7].

Prescriptive standards are problematic for two reasons. First, they present a safety problem: there is not particularly strong reason to believe that merely following any given process to the letter will necessarily lead to a safe system.

Second, they strongly restrict the technologies (e.g. algorithms and hardware) that can be certified as safe. This second problem would prevent the certification of the complex systems with which the CoSMoS project is concerned.

Because of the problems with prescriptive standards, there has been an increasing move towards *goal-based* safety standards. Such standards define the measure of safety that is required (for example, expected lives lost per operating hour), and the evidence of safety that must be supplied, then require system developers to present a structured argument that their system will meet the required level of safety. Rather than mandating processes or technology choices, goal-based standards set the bar for safety achievement and put the onus on system developers to provide evidence of safety. Provision of such evidence for complex systems is of obvious relevance to the CoSMoS project. Examples of goal-based standards include Def Stan 00-56 Issue 4 [6], which is particularly clear and straightforward.

Safety thinking, particularly in its goal-based form, is making inroads into other domains. For example, the ISO is developing an Assurance Case standard which will lay down standards for arguing non-safety properties of software systems. In the long term, it may be that standards for arguing dependability (the umbrella term that includes safety, security and other attributes – see Despotou in [8]) become widespread across many engineering domains.

3 The Basics of Safety Engineering

In this section, we give an overview of how a system developer can perform safety engineering and end up with a system that is safe to operate. It is written with the assumption that there is a goal-based safety standard in place that the developer must meet, but we feel that this overall approach is a strong one even where there is no legal requirement to follow it. We can only give a very general outline here – anyone performing safety engineering will need to use (or develop) a lot of other domain-specific guidance.

3.1 Hazards and Risk

At the very core of safety engineering is concept of the *hazard*. In its simplest sense, a hazard is a state which it is dangerous for the system to be in. Def Stan 00-56 defines a hazard as *"A physical situation or state of a system, often following from some initiating event, that may lead to an accident."* [6].

An example hazard for a car is *"Loss of steering control"*. An example for a chemical plant is *"Tank temperature exceeds flash point of contents"*. There is no hard rule for when one hazard is sufficiently distinct from another – hazards are a view that we impose on a system in order in order to manage the huge numbers of possible accidents and causes of those accidents (see Figure 1 for an illustration of this).

The process of identifying hazards is called *hazard identification*, and the process of deriving causes and consequences (accidents) for hazards is known

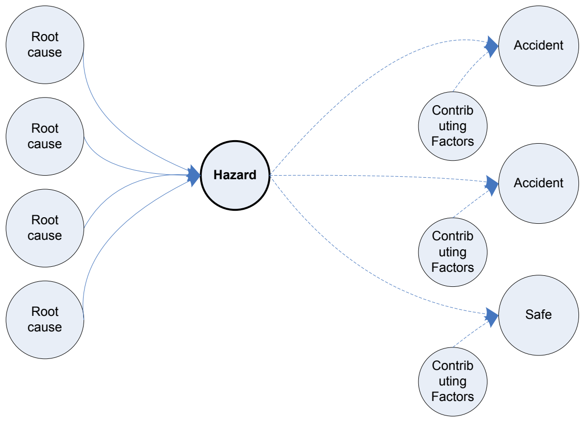


Fig. 1. Causes, Hazards and Accidents (reproduced from [9])

as *hazard analysis*. These activities need to happen early in any safety-critical engineering project, and should be updated throughout the lifecycle (even during operation). A variety of techniques exist for guiding engineers through this process, such as Functional Failure Analysis (FFA) (see Pumfrey in [9]) and HAZOP (see Kletz in [10]).

3.2 Predictive Analysis

Once hazards have been identified, we need to determine how often they will occur and what (how severe) the consequences will be if they do. The severity of each hazard needs to be measured in some standard way – several classification systems exist, but many can be seen in terms of mathematical expectation of loss of life (or a combination of that and the expectation of monetary cost). The combination of probability and severity is known as the *risk* posed by the hazard, and the combined risk from all hazards provides the *total system risk*.

A convincing prediction of hazard probability and severity is critical for safety certification. As noted in Section 2, certification requires that a developer show that the risk posed by the system is acceptable, and prediction of hazard risk is the centre of that.

3.3 Safety Requirements

In many cases, the inherent hazards in a system will make it unacceptably dangerous in its "naked man" (no safety measures) form. From hazard analysis, engineers will have to derive a number of additional system requirements that, if met, will reduce the risk posed by a hazard. These *safety requirements* then drive later safety engineering activity, and obviously have knock-on effects as they interact with requirements from other sources (e.g. the system's basic functional and performance requirements).

Safety requirements may be qualitative (*"If a wheel locks, then the anti-lock braking system must engage"*) or may be quantitative (*"The probability of a pressure relief valve jamming shut must be no more than* 1×10^{-4} *per operating hour"*).

In a goal-based safety regime, how safety requirements are met is up to the developer. The key, however, is that the developer must provide evidence that the requirements are adequate (that they really will provide the desired increase in safety) and that they have actually been achieved in the implemented system. The level of confidence needed will depend on the level of risk posed by the system's identified hazards – more dangerous systems require correspondingly better evidence.

Even if a requirement is actually adequate and is actually met (from the perspective of some hypothetical all-seeing observer), it may still be that the system developer cannot convincingly show this. This is particularly likely for novel systems and novel technologies. The simulation and analysis methods proposed by the CoSMoS project are clearly relevant here.

3.4 The Safety Case

A safety case is a document that presents a compelling structured argument that a system is safe, for some explicit definition of "safe". It is on the basis of the safety case that a regulator (or other independent safety assessor) approves a system for safety certification. For example, the required standard of safety might be expressed in terms of "Expected loss of life per operating hour".

There are several ways to structure and express a safety case. A plain natural language presentation can be used, or a tabular format, although there is increasing use of explicit argument structure notations such as the Goal Structuring Notation (GSN) (see Kelly in [11]).

Below the level of notation, a safety case must be organised in some systematic way so that the completeness and adequacy of the argument can be assessed. Several different *safety argument patterns* are presented in [11]. Perhaps the simplest structure is a breakdown by identified hazards – such a case would argue that all hazards had been identified, that all hazards had been adequately mitigated, and that the combined risk from all mitigated hazards was acceptable. The rest of this paper will assume such a structure.

The standard of argument in a safety case is *informal*, in the manner described by Toulmin in [12]. Formal notations and formal proof are emphatically *not* required – indeed, we are a very long way from a formal language that could adequately express all the diverse aspects of a safety case. Rather than proof, a safety case seeks to establish adequate *confidence* that the safety argument is sound. Specifically, it needs to provide evidence that the safety requirements derived from hazard analysis are met.

The level of confidence required for any particular claim depends on the system and the claim. Def Stan 00-56 makes this explicit: the required confidence depends on the risk posed by the system, its complexity and the degree of novelty (either in terms of technology or application). Specifically, it states: *"The* *quantity and quality of the evidence shall be commensurate with the potential risk posed by the system and the complexity of the system"* [6] – the novelty issue is presented in an additional diagram (the "McDermid Square"). "Complexity" here is not in the CoSMoS sense – rather, it means merely "complicatedness". (In any case, this is contingent on the developer's ability to understand and manipulate the system; it is in some sense "analysability".)

One part of any safety case will be the claim that the derived safety requirements are adequately complete. Typically, the developer will assert that all hazards have been identified, and that the set of requirements derived from each is adequate. Such claims can be difficult because they involve assertions about the (absence of) the unknown. They are, nevertheless, absolutely essential – if there is a missing hazard (or a missing cause of a known hazard) then the estimated total system risk could be very wrong.

In particular, a simple claim of adequate *effort* in hazard analysis is not sufficient – if the system being argued about is extremely complicated, and uses technologies that are poorly understood, then it may not be possible to make an adequate completeness claim. It may, therefore, not be possibly to certify the system – the developer may have to go back to the drawing board, changing the design or concept until it is tractable to analysis.

3.5 Operational Safety Management

Once a system has a safety case, has been certified, and is in active use, it still needs active management if it is to remain safe. The safety level of the system needs to be monitored and maintained. This *operational safety management* is critical because this is the point where prior safety-related modelling and analysis encounters the reality of the system in its real operating environment. The practical evidence coming from real operation has far higher confidence than the evidence from prior models or pre-operation tests.

System operators can and should track the accident and incident rates for the system. This can be used to estimate the actual safety risk of the system. Such *safety performance management* allows us to assess whether the risk levels claimed in the safety case are accurate. When a discrepancy is found, the developer must re-assess the safety of the overall system in the light of this change, and may need to change the design or operational procedures to restore an adequate level of safety. When this happens, the safety case should be updated; a safety case should be a *living* document.

3.6 Conservatism

A common theme throughout safety engineering (and indeed, throughout all safety-critical industries) is *conservatism*. Safety-critical systems are built using well-established technologies, methods and tools, often after similar non-safety developers have moved on. System developers are often extremely reluctant to change their development processes, architectural assumptions or even programming languages. In some sectors it is difficult or impossible to certify systems using technologies that have been accepted for decades in non-safety domains (for example, it is currently difficult to use software control in a UK nuclear energy system because the regulators will not accept it).

Conservatism in terms of technology is often motivated by the practicality of convincing analysis or by perceived protection from human error. Determinism is also a factor – if it is hard for an engineer to predict the precise behaviour of a system component at a critical time, then it is hard for them to claim that it meets any associated safety requirement.

As an example of conservatism and its motivation, consider memory management in software. Many software developers working in non-safety industries use languages such as Java and platforms such as the Java Virtual Machine that provide garbage collection, and therefore automatically reclaim dynamically-allocated memory. By contrast, software developers who are working on safety-critical software are generally unable to dynamically allocate memory at all (it is prohibited by the MISRA C subset [13] and simply not supported by the SPARK language [14].) They are, in a sense, two whole paradigms behind.

It is *not* true that dynamic memory allocation is fundamentally incompatible with safety. It is, however, a notorious source of programmer errors and very difficult to statically analyse. In particular, it is difficult to assert that the memory needed by the software will never exceed the amount that is available. By contrast, if a software program only uses static memory allocation, it is easy to determine how much memory is required.

Garbage collection provides some protection from programmer error (the classic case of losing all pointers to an allocated memory block before releasing it, thereby causing an unrecoverable "memory leak") but does nothing to bound the worst-case memory usage of a program. In addition to this, most garbage collection schemes are non-deterministic; it is impossible to predict exactly when the garbage collector will be called or how long it will take.

Conservatism will be a major obstacle to the creation of safety-critical complex systems. This will be both in its intuitive, "gut-feeling" form (complex systems are strange architectures implemented using strange algorithms on strange substrates) and in its well-reasoned form (if we can't predict the worst case behaviour of a particular emergent algorithm, then we may be forced to eschew it in favour of a conventional, centralised algorithm that we *can* understand – even if the average performance is much worse).

4 Safety of Complex Systems

To illustrate some safety activities in the context of a complex system, we will use the Artificial Platelet (AP) system. The AP system was used as a case study by the TUNA project (see [15] and [16]). It provides a set of artificial platelets that can be injected into a patient's blood stream to speed up wound clotting (for example after a serious accident). Obviously, this has inherent safety risk, so safety engineering is essential.

Section 4.1 gives an initial identification of accidents and hazards for the AP system concept, and Section 4.2 discusses the architectural implications of some of these hazards. Section 4.3 then shows how a safety case could be developed from this starting point.

4.1 Initial Accident and Hazard Identification for the AP Concept

As noted in Section 3.4, it is critical that safety engineers identify all the hazards that a system can exhibit. The set of hazards is unique to the particular system being developed – it may not be shared with other systems using the same technologies or approaches.

For well-established types of systems (such as cars and chemical plants) the types of accidents that can occur are well understood (e.g. crashes and explosive reactions). We can therefore move straight to indentifying hazards. The AP system is highly novel, however, and the set of possible accidents is *not* obvious. We must, therefore, identify a set of accident scenarios before we can identify all hazards.

Tables 1 and 2 show the output of an initial accident and hazard identification for the AP concept, developed by brainstorming between the authors (two academics specialising in safety engineering and a medical doctor). This method (safety engineers brainstorming with domain experts) is a common and important way to start the identification of hazards.

The accident scenarios presented in Table 1 are events occuring at a relatively large scale, and are events that we care about directly (they are events that have very direct consequences for the survival and future health of patients). The hazards in Table 2 are events at the level of the AP system that are not necessarily of immediate concern, but which could lead to one or more of the identified accident scenarios. In particular, they could lead to accidents *without anything else going wrong*. Note how one hazard may lead to multiple accident scenarios. Indeed, it is often possible to group many similar accidents under a smaller number of hazards; this may help to keep the safety case manageable (this was shown diagramatically in Figure 1).

A note on immune response: Hazard H2 and accident scenario A5 may be caused by an immune response, and because the immune system learns this may become worse over time. If an immune response did occur, the first application of AP to a given patient would have no visible ill effects, while the second might have very severe ones.

It is important to remember that safety engineers aren't very worried about hazards that *always* happen – such hazards will become obvious before long. Their prime concern is with hazards that will rarely manifest, but not so rarely as to be of no concern. A common baseline (in process plants and aviation) is that all hazards combined should cause a fatality no more than once every 10^6 years, but this varies widely.

One important variant of this is finding hazards that will manifest often under certain circumstances – in the AP case, this might mean only for some

Table 1. Identified AP Accident Scenarios

ID	**Description**	**Notes / Causes & Consequences**
A1	AP fail to form clot	If not announced may delay other treatment.
A2	Clot is too short-lived	Human platelet transfusions have a very limited lifespan and are often used in people who have low platelets who are actively bleeding, or alternatively, immediately prior to an operation. Temporary clotting is of little value and AP would need to at least match the current human platelet lifespan.
A3	Clot blocks blood vessel	Could cause stroke, coronary thrombosis or potentially a pulmonary embolism. Announcement required at *inter-platelet/multi-agent* level.
A4	AP cause Disseminated Intravascular Coagulation (DIC) – widespread clots form in multiple vessels	Usually fatal. Very poorly understood phenomena – root cause is not known, but platelets are involved in the mechanism. Unregulated clotting occurs, which leads to haemorrhage as clotting factors and platelets are subsequently used up.
A5	Allergic reaction to AP	May be immediate or delayed. Effect could range from minor symptom (e.g. rash) to fatal anaphylaxis.
A6	AP act as infection vector	Particularly dangerous given that patient is already in poor health.
A7	AP damage permeable membrane	AP may interact differently in response to the body's permeable membranes such as in the kidneys filtering system. It could cause an obstruction or pass through the kidney filtering system inappropriately. This could damage the filtering apparatus and lead to renal impairment or even organ failure.
A8	AP prevent secondary haemostasis	May prevent formation of stable, long-term clot.

Table 2. Identified AP Hazards

ID	Description	Notes / Causes & Consequences
H1	AP go somewhere they shouldn't	e.g. crossing the blood-brain barrier. May cause A3 or A7.
H2	AP destroyed by immune system	May cause A1 or A2. Detection and announcement may be difficult.
H3	AP lifespan too short	May cause A2.
H4	AP form oversized clot	May cause A3.
H5	AP contaminated with infectious agent	May cause A6. May depend on platelet storage arrangement – might need to be stored at or near body temperature. Bacterial infection most likely in this scenario.
H6	AP have unexpected interaction with permeable membrane	i.e. their interaction is unlike that of natural platelets. May cause A7.
H7	AP fail to release mediators for secondary haemostasis	May cause A8 (see clotting cascade diagram in Figure 2).

patients, only when administered in combinations with some drugs, or only for patients with some other medical condition.

H5 (AP contaminated with infectious agent) shows how performing safety analysis early can save effort (and maybe save your engineering project). If you know that preventing bacterial growth on AP (or in their storage medium) is critical, then you can take steps to achieve this. If you *don't* know that this is critical until late in the project, then you might have made design decisions that conflict with this. For example, your AP might have to be stored at human body temperature (which is ideal for bacterial growth).

It is important to remember that you don't have to argue safety *in all possible situations*, provided that you can argue that you know when your system is *not* safe to use. Taking A3 as an example, the AP system might not be safe to use on a patient with a cardiac stent, because the stent may be a likely site for an unwanted artificial clot. If you know this, you can explicitly exclude this situation from the safety case (except to note that this restriction will be clearly documented).

Tables 1 and 2 are, of course, only a starting point, but they illustrates the typical form and character of a high-level hazard analysis. It is likely that there are possible accidents that are not covered by Table 1. There are certainly ways that those accidents could occur that are not covered by the hazards in Table 2 (for example, no hazard is identified that could lead to accident scenario A4). During the development of the AP system, the hazard list would be expanded and additional detail would be added. Further brainstorming with other domain experts (and more specialised experts) would be valuable (the Oxford Handbook of Acute Medicine [18] gives "abnormal platelet behaviour" as a possible cause for many different conditions, and various platelet-related disorders are discussed in [19]).

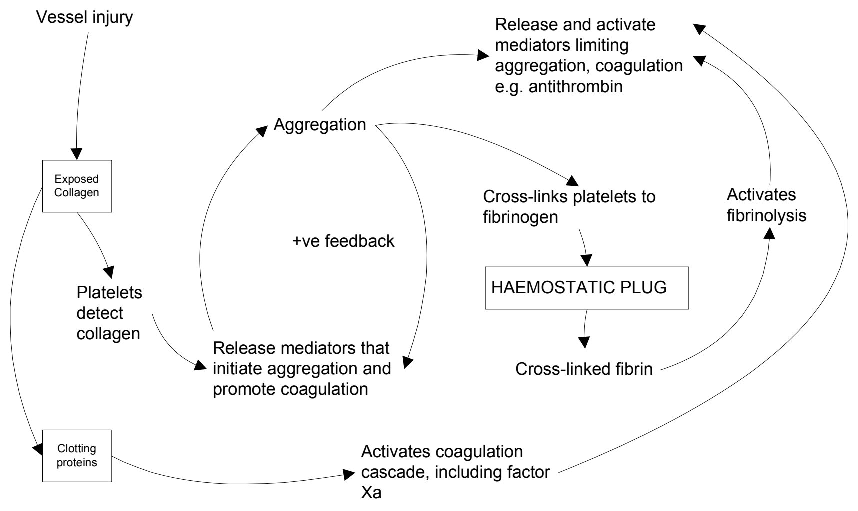

Vessel injury
Release and activate mediators limiting aggregation, coagulation e.g. antithrombin
Aggregation
Exposed Collagen
Cross-links platelets to fibrinogen
Activates fibrinolysis
+ve feedback
Platelets detect collagen
HAEMOSTATIC PLUG
Release mediators that initiate aggregation and promote coagulation
Cross-linked fibrin
Clotting proteins
Activates coagulation cascade, including factor Xa

Fig. 2. Clotting Cascade (after [17])

As more sophisticated models became available, engineers would use a variety of hazard analysis techniques that work over these models. Some general-purpose methods were identified in Section 3.1, and Section 5 outlines a possible simulation-based technique. For example, the HAZOP technique (see [10]) is widely used for analysing process diagrams in chemical plants. It could be adapted for use on biological process diagrams, such as the clotting cascade shown in Figure 2. (It should be noted that Figure 2 is a simplified representation of the cascade, only touching on the secondary (non-platelet) part; more detailed representations exist.)

One corollary of the above hazard analysis is that the safety of the AP system has a tremendous dependency on the behaviour of the patient's immune system. Any simulation-based attempt to address these hazards will need to be paired with a reasonable immune system model. Most likely, this environment model will be built up over time (e.g. through concept, design, animal trials and human trials). Scope control is crucial here – developers may need to restrict the use of simulation to where it is explicitly needed (e.g. for "emergent hazards").

In some ways, the AP system is an easy example because we already have natural platelets to learn from. If we were considering an entirely novel bloodstream-dwelling nanomachine, we'd have a more difficult job. For example, consider whether the hazards presented by an artificial heart are the same as those of a natural heart – there is probably significant overlap (particular those hazards concerned with the blood vessels around the heart) but the artificial heart has a very different internal mechanism.

4.2 Individual and Multi-agent Hazards

H1 (AP in wrong location) could potentially be detected by an *individual* AP – the AP to detect that the surrounding tissue is of the wrong type. By contrast, hazard H4 (AP form oversized clot) is a state at the *inter-platelet* (i.e. *multi-agent*) level, and can be detected only by something that has an overall awareness of multiple APs.

This is an architectural distinction – it is much easier to argue that a hazard is mitigated if it can be resolved at the individual platelet level. For example, an AP could be built to detect whether it was in or near a blood vessel, and to self-destruct if it was not. Alternatively, it might be possible to argue that the lifespan of an AP, although long enough to be useful for clotting, is not long enough to cause problems in other tissue. On a theoretical level, these kinds of arguments are straightforward – they're much like the ones we use in existing safety-critical systems.

By contrast, it is difficult to achieve and argue mitigation for multi-agent hazards. Most likely, an engineer would need to argue that the platelet would be able to detect that the AP system had formed a clot where it was not needed, and then disassemble it. This is an extra requirement for emergent behaviour (extra to those needed to provide useful clotting in the first place). If this approach was taken, this would be an example of a *safety requirement* as introduced in Section 3.3.

(One alternative might be to detect unwanted clots through external imaging, but this would raise a variety of issues including imaging equipment availability, interaction of imaging actions with other treatment of a critically injured patient, and human factors such as ability to see small developing clots in obscure locations).

'Detect' in the above paragraphs could have a number of meanings. One is that the AP has some kind of explicit knowledge of its state (and can therefore take an explicit action) – this is the norm for conventional systems. Another is that the AP is engineered so as to only be stable under a narrow set of 'safe' conditions. For example, it might be that the AP could be designed only to be stable in the physical and chemical environment of a suitable blood vessel – if it was in any other location, it would break up and become inert. For ultra-simple (e.g. nanoscale-assembled) agents the latter approach may be the only viable one.

Nanoscale devices may be novel in that they are built on a single substrate, without the traditional divisions between e.g. software and hardware in a conventional vehicle or robot. They may be built from a single material, both logic and actuators, making them closer to a clockwork device than an electronic robot. In safety engineering, we often rely on the software-hardware divide to manage and contain the software contribution to hazards. More generally, this illustrates how novel technology may break existing safety techniques.

4.3 Building a Safety Case for the AP System

A common and practical approach to building safety cases is to argue that all hazards presented by the system present an acceptably low risk. Figure 4 shows the top level of a possible safety case for the AP system, presented in the Goal Structuring Notation (GSN).

Arguments in GSN have a hierarchical structure. At the top of the structure is a *goal* which takes the form of a particular claim about the system (in Figure 4 this is the node RisksTolerable). The level below that breaks this down into multiple child goals. Each of these child goals then broken down in turn, until we reach a point where the goals can be supported by ("solved by") explicit *evidence* (evidence is shown in a GSN diagram as a "Solution" node). Between any goal and its child goals a *strategy* may be used; this explains how the breakdown is being performed (in Figure 4, ArgOverHazards shows that we are claiming that the risk from all hazards is tolerable on the basis that the risk associated with each identified hazard is tolerable). Figure 3 gives a key to the symbols used in GSN; for a more comprehensive description of the notation, see [11] or [20].

(Note: The strategy ArgOverHazards breaks down the argument in terms of the top-level hazards that were identified in Table 2).

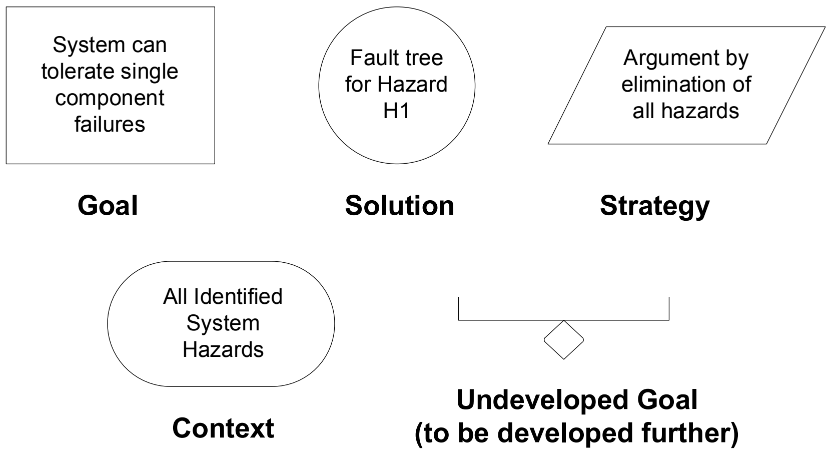
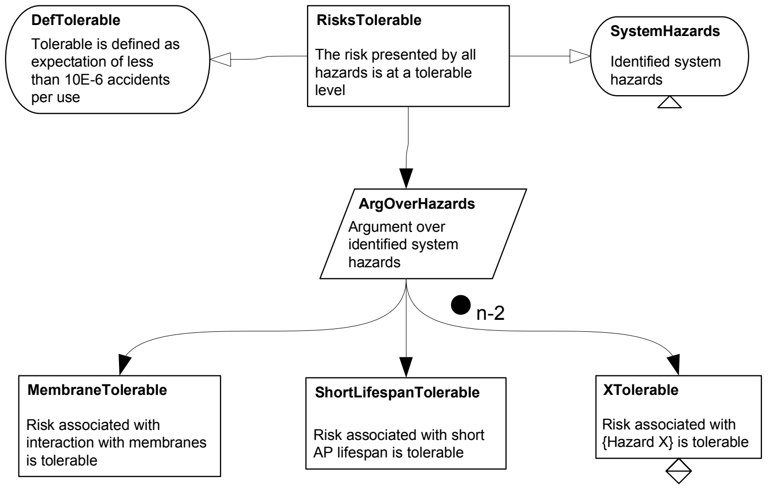

System can tolerate single component failures
Goal
Fault tree for Hazard H1
Solution
Argument by elimination of all hazards
Strategy
All Identified System Hazards
Context
Undeveloped Goal (to be developed further)

Fig. 3. Principal Elements of the Goal Structuring Notation (from [20])

The context node DefTolerable is very important – it defines the term 'tolerable' that is used in the rest of the argument. The definition used here, "expectation of less than 10^{-6} accidents per use" is arbitrary, although not untypical of existing safety arguments. If we knew what the regulator in this case would require as a standard of safety, then we can change our definition (although we might then need to change our argument structure in order to meet it).



DefTolerable
Tolerable is defined as expectation of less than 10E-6 accidents per use
RisksTolerable
The risk presented by all hazards is at a tolerable level
SystemHazards
Identified system hazards
ArgOverHazards
Argument over identified system hazards
n-2
MembraneTolerable
Risk associated with interaction with membranes is tolerable
ShortLifespanTolerable
Risk associated with short AP lifespan is tolerable
XTolerable
Risk associated with {Hazard X} is tolerable

Fig. 4. Top Level of AP Safety Case

It is important to note that the definition of 'tolerable' provided by DefTolerable applies to the whole AP system active within a particular patient; it sets a probablistic tolerability for accidents at the macro level. This may eventually be converted into probablistic requirements on behaviour at the *micro* level in terms of individual AP. The value of the probabilites used at that level would depend on the macro-level tolerability, the number of AP likely to be given in a single treatment (for AP, that could be billions) and the ability of the overall AP system to mitigate undesired behaviour by a small number of AP. No simple mapping from macro-level accident rates to micro-level failure or hazard rates is possible.

The big advantage of arguing over hazards, compared to other ways of structuring a safety case, is that we can adopt a unique argument structure for claiming that each hazard is tolerable. Figure 6 illustrates this for hazard H6 – we argue over the various membranes that the AP could encounter, attempting to show that the interaction with each will be safe. Similarly, Figure 7 shows the argument for hazard H3 – in this case, we argue across the normal and abnormal cases, then (in Figure 8) over the possible causes of the abnormal case.

Notice how Figure 8 ends with undeveloped goals (indicated by the diamonds underneath them). Before this safety case could even potentially be accepted, the developer would need to decompose it further until they reached solutions (shown as circles in e.g. Figure 6).

In a real safety case, parallel arguments would be needed for the completeness of hazard and cause identification, and about the combined risk from all hazards. The latter is straightforward but the former can be very hard – see Section 5.5. There would probably, also, need to be more context and strategy nodes.

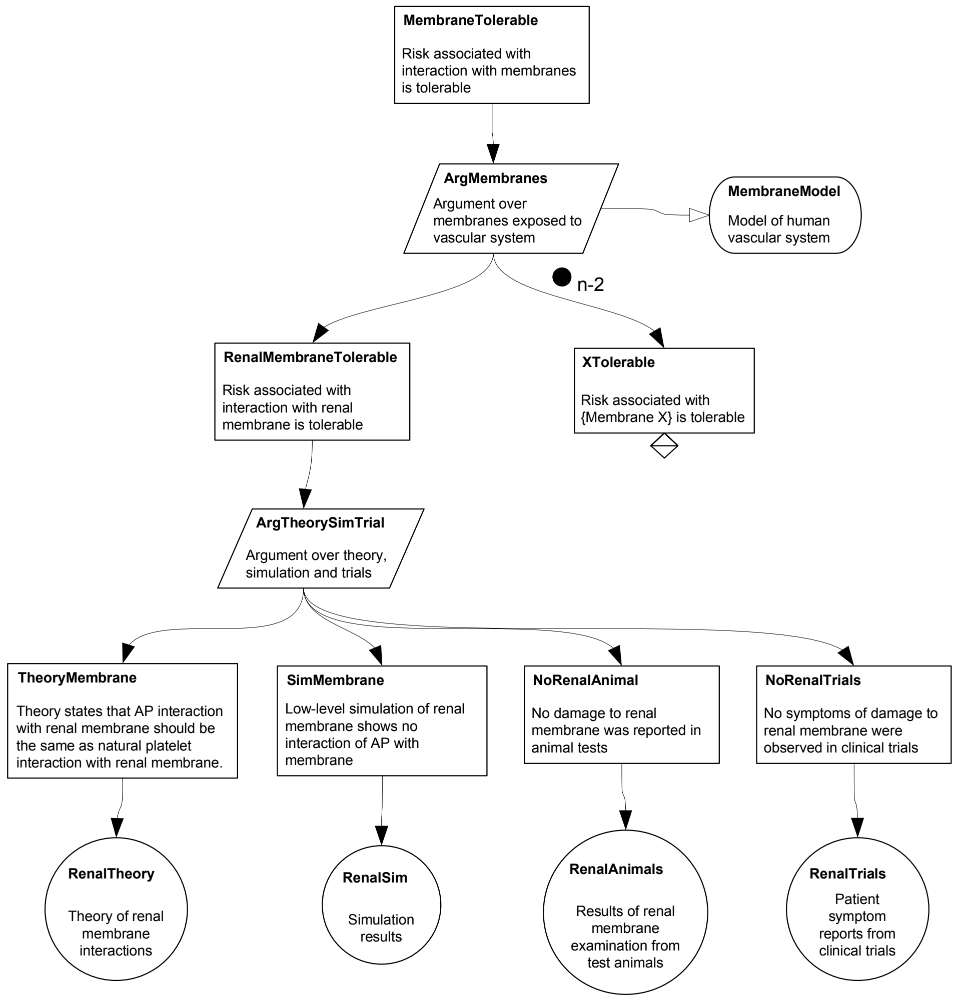

MembraneTolerable
Risk associated with interaction with membranes is tolerable
ArgMembranes
Argument over membranes exposed to vascular system
MembraneModel
Model of human vascular system
n-2
RenalMembraneTolerable
Risk associated with interaction with renal membrane is tolerable
XTolerable
Risk associated with {Membrane X} is tolerable
ArgTheorySimTrial
Argument over theory, simulation and trials
TheoryMembrane
Theory states that AP interaction with renal membrane should be the same as natural platelet interaction with renal membrane.
SimMembrane
Low-level simulation of renal membrane shows no interaction of AP with membrane
NoRenalAnimal
No damage to renal membrane was reported in animal tests
NoRenalTrials
No symptoms of damage to renal membrane were observed in clinical trials
RenalTheory
Theory of renal membrane interactions
RenalSim
Simulation results
RenalAnimals
Results of renal membrane examination from test animals
RenalTrials
Patient symptom reports from clinical trials

Fig. 5. Argument over Membranes

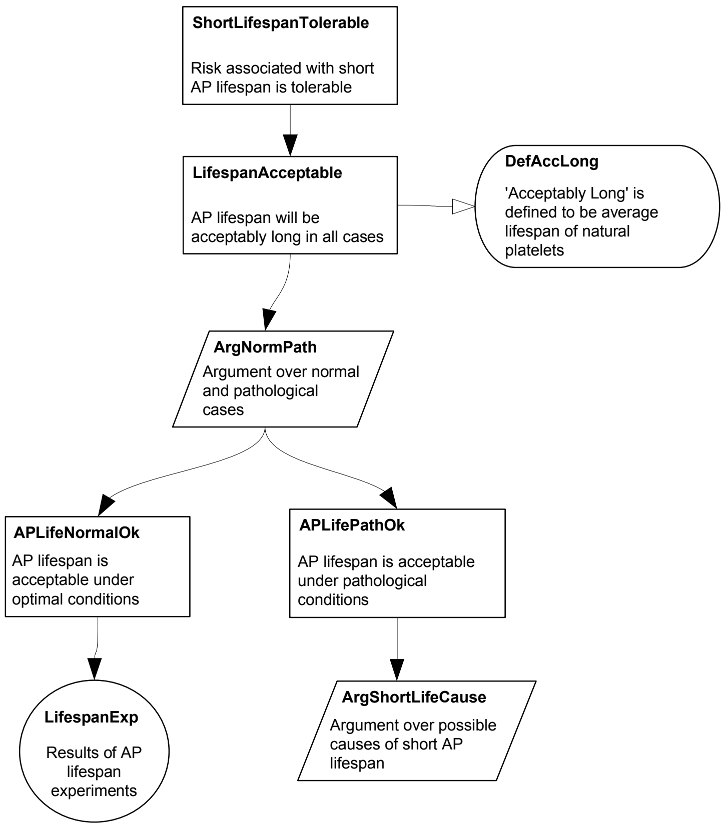

ShortLifespanTolerable
Risk associated with short AP lifespan is tolerable
LifespanAcceptable
AP lifespan will be acceptably long in all cases
DefAccLong
'Acceptably Long' is defined to be average lifespan of natural platelets
ArgNormPath
Argument over normal and pathological cases
APLifeNormalOk
AP lifespan is acceptable under optimal conditions
APLifePathOk
AP lifespan is acceptable under pathological conditions
LifespanExp
Results of AP lifespan experiments
ArgShortLifeCause
Argument over possible causes of short AP lifespan

Fig. 6. Argument for Short Lifespan Hazard

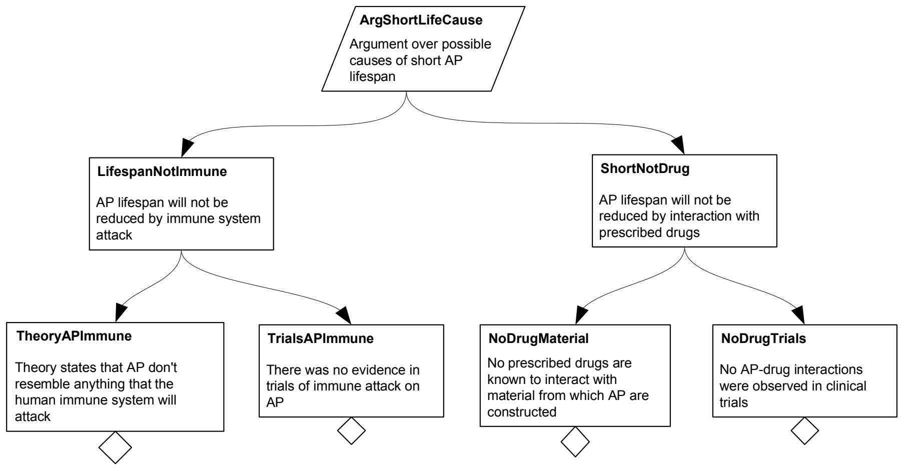

ArgShortLifeCause
Argument over possible causes of short AP lifespan
LifespanNotImmune
AP lifespan will not be reduced by immune system attack
ShortNotDrug
AP lifespan will not be reduced by interaction with prescribed drugs
TheoryAPImmune
Theory states that AP don't resemble anything that the human immune system will attack
TrialsAPImmune
There was no evidence in trials of immune attack on AP
NoDrugMaterial
No prescribed drugs are known to interact with material from which AP are constructed
NoDrugTrials
No AP-drug interactions were observed in clinical trials

Fig. 7. Argument About Causes of Abnormally Short Lifespan

The ‘trick’ exploited by this safety case structure is that it deals with as many hazards as possible by simple means. Neither of the hazard arguments shown involves appeal to complex emergent properties – instead, we have dealt with two hazards in relatively simple terms. This allows us to focus our effort on where it is most needed (on the complex, emergent accident hazards like H4). Complex hazards will require the development of new techniques and patterns. Some progress towards this has been made in unpublished work by Polack [21].

In the argument fragments above, the argument’s legs are combined informally, and would rely on expert judgement to assess the overall contribution of evidence to the top goal. The general intuition is that when there is a goal solved by N child goals, then adding an additional diverse child goal will increase the confidence in the parent. There is no accepted, explicit method of assessing and combining argument confidence, although [22] and [23] comment on what engineers seem to do in practice. (In particular, Littlewood and Wright show in [23] how the “extra leg” intuition may be misleading in some cases).

To adequately assess the strength of complex arguments, a systematic method for propagating confidence through the argument is desirable. Weaver, in [24], presents a system of Safety Assurance Levels (SALs) for performing this propagation. Each low-level goal is assigned a SAL (from 1 to 4) and these propagate up through the argument until the top goal is reached. Independent child goals may give their parent a SAL that is higher than either of them; interdependent child goals will give their parent a SAL that is the lowest of all of them. The weakness of SALs is that it is not clear what a given SAL means in tangible terms (if a goal has SAL 2, what exactly does that mean about the system being argued about?).

A mechanism for assigning truly quantitative confidence in useful real-world terms would be very valuable. Littlewood, in [25], discusses how such a mechanism might be found in Bayesian Belief Nets, which can express structural relationships between beliefs in terms of probability, although he also notes that there are a number of unsolved problems with the approach.

5 Simulation in Safety Engineering – An Example

Two of the authors previously developed a simulation-based approach to hazard analysis for complex systems-of-systems (SoS). The work, carried out as part of the HIRTS DARP project, was in response to increasing integration of large-scale SoS such as network-enabled military units and Air Traffic Control (ATC). An overview of the work follows; for more detail, see Alexander [26].

The motivating concern for the work was that large-scale SoS are not very tractable to conventional hazard identification and analysis techniques. The SoS of concern are complex (in the sense used by CoSMoS), immensely complicated, distributed, and very heterogeneous. Our scope was limited to hazard identification and analysis – we didn’t attempt to provide confirmatory safety analysis (this has certain implications, which are discussed in Section 5.5, below).

Merely identifying and explaining hazards is difficult in SoS. If we have a failure in one entity, what is the consequence at the SoS level. For example, if a reconnaissance aircraft in a military unit spots a civilian car and believes it is an enemy tank, what happens to the car? The consequence depends on the capabilities of the other entities in the SoS, the operational procedures in force, and many properties of the SoS's dynamic state (right down to the psychological disposition of the humans involved: are they relaxed or on edge? Are they expecting the enemy or do they think there's a ceasefire? Do they *want* to see the enemy?) Timing may be a critical factor (when does the misidentification occur, with respect to the patrol times of other reconnaissance entities?).

It may be, in a complex SoS, that we don't even need a "failure" in order for an accident to occur. It may be that under certain circumstances, under certain system states, normal behaviour by all the entities involved is enough to cause an accident. This could be seen as an example of "negative emergence". The enormous state space of an SoS makes these dangerous states, and the paths to them, difficult to discover.

This leads to the concept of an *SoS Hazard* – a state of an SoS. Taking the Def Stan 00-56 definition of 'hazard' (from Section 3.1) we can define this as *"A condition of a SoS configuration, physical or otherwise, that can lead to an accident."* We can explicitly exclude from this those hazards that are confined to a single entity – states where an entity has suffered a failure and may go on to directly cause an accident (for example, an aircraft suffers an engine failure and crashes into the ground). These single entity hazards are relatively tractable using current techniques. It is the multiple-entity hazards that are challenging. (A similar, but more general, concept of *complex system hazard* could be defined). The aim of the work discussed in this section is to find and explain SoS hazards.

5.1 Method Overview

The method requires, first, that the SoS safety team develop a multi-agent model of the SoS. They do this by taking an appropriate source model (such as a description of the system in the Ministry of Defence's MODAF notation [27]) and identifying specific safety concerns that they need to model (such as collisions between aircraft). They must also identify (a) the vignettes that the SoS will be expected to participate in and (b) a set of reasonable deviations that may occur in practice, such as a system suffering a particular kind of failure. The resulting multi-agent model must be implemented in a multi-agent simulation framework, thereby making the model executable.

Once an executable model is available, the 'space' represented by the deviations of that model must be explored. This is performed by running the model with different combinations of deviations and observing the results.

As each run executes, the actions and states of the system components are logged so that they can be studied later. This invariably produces a huge volume of output. To aid comprehension of this data, machine learning techniques can then be used to extract high-level descriptions of hazards. These descriptions are relationships between certain combinations of deviations and certain accident events. Once interesting accident runs are identified, causal explanations can be derived using an agent tracing tool. Engineers can then use the explanations to study the plausibility of the simulated accidents, and modify the SoS configuration or operating arrangements in light of the more credible relationships.

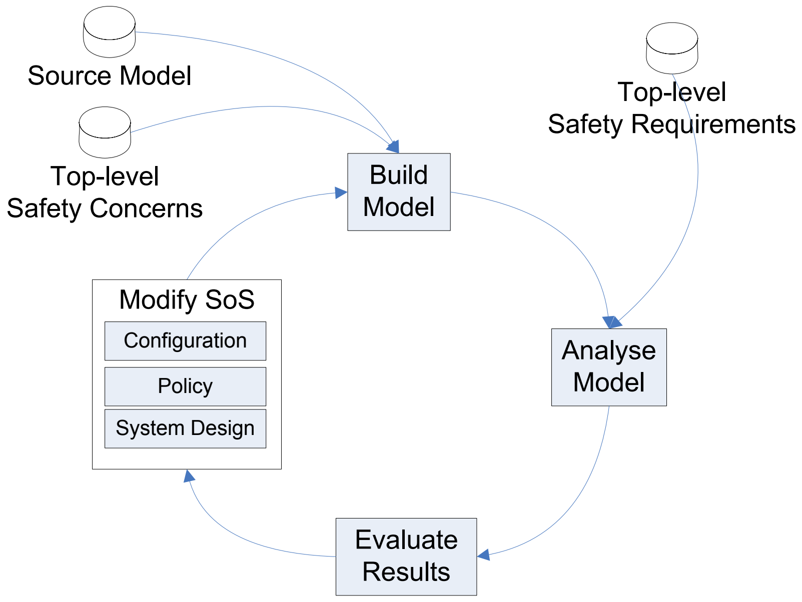


Fig. 8. Overview of the SoS Hazard Analysis Process

An outline of this process is shown in Figure 8, and key aspects are expanded on in the following sections.

5.2 Modelling Approach

A common concern in simulation modelling is that, in going from a paper model to an implemented simulation, distortions and errors can be introduced. To help alleviate this, we adopted the concept of explicit *concerns*. These concerns were initially expressed in terms of aspects of the source model, and then updated and checked at each modelling stage or iteration of the model. These concerns could be either deviations or accidents, and provided a starting point in terms of causes and their eventual consequences. The rest of the process then works from there to derive the intermediate paths and states that linked them.

In addition to deviations coming from the concerns, we provided a method for deriving agent deviations by applying a set of five guide words to a six-part breakdown of a generic agent model. This is similar to the existing methods FFA and HAZOPS mentioned in Section 3.1.

The process we used for developing the agent model was an adaptation of the Prometheus process (see [28] for a description). Although this was not a true refinement method, requiring considerable engineer input, it structured the process and provided a range of cross-checks between the several stages of model development. For example, Prometheus leads developers to describe inter-agent communication in terms of high-level protocols before the agents themselves are developed in detail. Once the agents *are* developed, the Prometheus Design Tool (described in [29]) will automatically check that the agents have the message types defined that they need in order implement the protocol. (It cannot, however, verify that they will restrict their message sequences to those allowed by the protocol.)

The CoSMoS project aims to provide methods for developing high-quality declarative models (rather than the partial MODAF models that we worked with) and offers the prospect of more systematic refinement from model to simulation. However, even if we had been able to perform a true refinement from source model to implementation, we still would not be able to assume that the simulation represents the real world. For our explicit purposes (hazard analysis only) this is not necessarily a problem, but we cannot escape it entirely. See Section 5.5 for further discussion of this issue.

5.3 Analysis Techniques

Once the model has been built, it must be analysed. The analysis technique must select simulation runs to be performed so as to achieve adequate coverage of the parameter space of the simulation while spending the minimum of computation time. We would like to exhaustively explore the parameter space demarcated by all agent deviations, because this would reveal all behaviour paths that were implemented by the simulation model. In practice, this will be impossible unless we use a toy example.

Our solution was to specify a probability for the occurrence of each deviation (in any given simulation run), and then perform runs only for those combinations where the combined probability is above a certain threshold. The simulation engine decides whether or not to run each combination by comparing its combined probability to a threshold for ‘incredibility of failure’. This concept originally stems from the nuclear industry – dangerous situations that appear to be more improbable than this threshold are not studied further in hazard analysis. A value for this is given in [30] as 10^{-7} per year of operation (equivalent to 10^{-11} per hour), and this value is adopted here.

Once those runs have been performed, the accidents that occurred need to be identified and their causes found. The former task is relatively easy, since the set of possible accidents is small. The latter, however, is harder, and machine learning techniques have been adopted to make it tractable.

For our purposes, the task of machine learning can be viewed as one of function approximation from a set of *training instances* expressed as input-output pairs; given a function specification (a set of named input parameters (the 'features' used for learning) and a particular form of output value), the algorithm learns the relationship between combinations of parameter values and the output of the target function for those values.

In our approach, the features represent parameters of the simulation and the output values are the consequences within the simulation. All the features used in the current work are deviations that are applied to the model, and the target function is the set of accidents that occurs during the simulation run. We used a decision-tree learning algorithm because it could learn from Boolean-valued parameters (deviations) and discrete-valued outputs (accidents), and present the resulting rules in human-readable form.

The output of the learning algorithm is a set of rules that describes the relationship between deviations and accidents. For example, a rule might be *"Aircraft 1 lost_radio_comms causes aircraft 1 to collide with aircraft 2"*. Such rules, however, only explain how accidents occur in very broad terms. In order to choose appropriate definitions of our hazards, or to take action to prevent or mitigate them, more detailed information about causation is required.

Lam and Barber, in [31] present a tool-supported approach to the comprehension of agent systems. Given a log of the events that occurred in a single simulation run and an event of interest within that run, the tool tries to explain *why* that event happened in terms of its immediate causes. Those causes can each then be explained in the same way, and the process repeated until the final explanation is in terms of the initial state of the simulation or 'external' events that occurred. This explanation, complete or partial, can be expressed as a causal graph leading to the event that we asked the tool to explain.

A simple example of such an explanation would be of the form *"UAV 1 received a percept indicating the location of an enemy unit. This caused it to form a goal of destroying that enemy unit, which it selected the 'air strike' plan to resolve, and as a consequence of that plan the UAV conducted the 'attack' action using a laser-guided bomb"*.

The tool achieves this by storing what Lam and Barber call 'background knowledge'. This is a set of possible causal relationships between events of different types and different properties. As the tool tries to explain each event, it reviews these rules to find which earlier events could have caused it.

Once the analysis is complete, the analyst must evaluate the significance of the results, in consultation with the rest of the project safety team and other stakeholders. It is likely, particularly early on in development, that the analysis will reveal problems with the SoS that need to be resolved. As indicated in Figure 8, this may involve changes to the configuration of the SoS, to the design of the individual elements, or to the operational safety policy under which the SoS operates.

5.4 Strengths

By using simulation, our approach is able to search a huge amount of the state space. By using probability as a heuristic for the selection of runs, we can prioritise our explanations towards those parts of the space that are most likely to occur in practice. The approach is particularly strong in this regard when compared to the manual techniques that have traditionally been used in this role.

Using simulation paired with machine learning and tracing (rather than an explicit exploration technique such as model-checking) allows engineers to use the approach with almost any kind of multi-agent model. For example, agent state may be modelled in terms of discrete states or continuous-value parameters, and agent behaviour may be defined by pre-written scripts or by a neural network.

The tracing tool provides a head start to analysts who want to explain how a particular accident happened. Simple learning of relationships between deviations and accidents may not be sufficient for human comprehension – in particular, it is difficult for humans to read extensive logs of simulation events. Animation can help here, but it is still difficult to see causes that are well-separated in terms of time.

The key value of the approach is that if it finds one hazard in an SoS that was not found by other means, then it is useful. This is true even if it also identifies a number of hazards that are initially plausible but turn out to be unrealistic. As it is not intended to provide confirmatory safety analysis, errors and omissions are acceptable as long as some hazards are realistic. (There could, of course, come a point where the cost of investigating false hazards made the approach impractical.)

In , we applied our approach to two plausible case studies (one of them purely hypothetical, the other provided by industry) and derived some interesting hazards. It is important that this be replicated in a real industrial or operational development, but these results are promising.

All of the above is likely to be true and relevant for the use of simulation in CoSMoS. It is important to note, though, that if simulation is to be used in direct support of a safety argument then it must aim higher than the work described in this section, particularly in terms of gaining confidence in the results (see 'Challenges', below).

5.5 Challenges

The approach described above raises a number of challenges. If the CoSMoS project uses simulation in a similar role, it will have to face these challenges.

The biggest concern with this work is the question of completeness – whether or not an adequately complete set of hazards, and causes of those hazards, has been identified. There are two aspects of this – completeness with respect to model (does the analysis reveal all the hazards in the simulated model), and completeness of the model with respect to reality (does the model implement, and reveal in simulation, all the hazards that exist in the real system). Some confidence that there are *no* unidentified serious hazards is vital for building a safety case. Ultimately, such a confidence cannot be empirically observed; it must come down to *theory*, theory about when it is reasonable to believe that your hazard analysis is complete.

A second concern is related – does the approach provide a false sense of security? Any safety technique or process will have an effect on the perceptions of the engineers involved. A highly detailed simulation model (especially when combined with an attractive animation) may convince engineers that lack of hazards is evidence of their absence. Put another way – any hazard identification or hazard analysis has an implicit, intuitive role in safety analysis. The key is that it should give confidence only in proportion to its adequacy.

A final concern is that the explanations produced by the tracing tool may be *too* compelling. If the tracer leads naturally to a plausible explanation of a mechanism by which a hazard can occur, they may cause engineers to ignore other alternative explanations.

6 The Challenges of Small-Scale Simulation

One can draw a distinction between large-scale simulation (such as the simulations of military units and air traffic discussed in Section 5) and small-scale simulation (such as the simulation of cells and nanites with which CoSMoS is primarily concerned). It could be suggested that in large-scale simulation the properties of the agents (including their local behaviour) are well understood, and that the role of simulation is to combine these to determine their emergent behaviour when they work together. In the small-scale case, the behaviour of the agents may be less well known, and the role of the simulation is to explain (and allow analysts to comprehend) the observed high-level behaviour in terms of low-level rules.

Alternatively, the distinction between large and small could be replaced by one between and engineering and scientific simulation. Engineering simulation is concerned with modelling well-understood parts in situations where their combined behaviour is too complex to understand; scientific simulation is concerned with understanding how parts give rise to well-understood high-level behaviour.

This second conceptualisation leaves potential for engineering simulation of small-scale entities. Can we, however, really know the fine detail of tiny things? For example, platelets may be relatively simple when compared to (e.g.) white blood cells, but they are still not completely understood. Some aspects of their high-level behaviour are also unexplained, such as the causes of the DIC phenomena mentioned in Table 1. The behaviour of entities such as platelets cannot be understood without reference to the environment that they operate in; for platelets this is, at the very least, the human vascular system. This creates a huge demand for knowledge, particularly for knowledge of combinatorial interactions between the diverse entities and processes in the system and the environment.

One can go further, however. Can we really know the precise local behaviour of large-scale entities such as vehicles? We certainly know gross detail (weight, dimension, maximum engine output), but as we move to progressively finer detail (mechanical tolerances, network protocols, presence of software faults) it becomes harder to fully describe, or indeed to know at all. Behaviour over the long term, as the system degrades through operation and is maintained, is particularly difficult to predict. Engineers have often been surprised by the behaviour of vehicles; for example, consider the 1974 air crash caused by the hard-to-shut cargo door on the DC-10 aircraft [32]. Safety engineering, when dealing with the levels of safety that we now expect in our society, deals with *tiny* probabilities.

It can be observed that, for real systems, the small-scale is always present *inside* the large scale. An aircraft may have a human pilot – the pilot contains blood which contains platelets. If we drill down further, into the individual platelets and (primarily) the mitochondria which they contain, then we have the respiratory cycle whereby they generate the power to operate. This cycle is invisible at the high level, but is intricate at the sub-cellular level; e.g. a diagram of the Krebs Cycle in [33] has 25 edges and 26 nodes (of which 17 are unique substance names).

The respiratory cycle is simple for the SoS engineer – at worst, they'll need to think of air, water and calories (e.g. blood sugar levels affect alertness). For the AP engineer, however, the respiratory cycle is close enough to be a concern – it is possible that AP will interact with the substances involved at a molecular level (e.g. it might be that AP attract and accumulate a respiratory by-product). This illustrates how *small-scale engineering dredges up fine detail.* As we move down into the micro-scale and nano-scale, phenomena that were manageable (indeed, barely of concern) in the large become critically important.

Ultimately, however, it is the high level that matters. For safety engineering, this means that we need adequately compelling evidence that the identified safety requirements are met (at the level of the whole system). For those properties of the system that are not relevant to any safety requirement, we do not need any evidence. Of course, there may need to be an argument made that these properties are truly irrelevant. Put another way - we do not need accurate models of the whole system. We *do* need evidence that we can predict certain key properties, and that the values of those properties will be suitable. And we can constrain the situations under which we need to predict these properties by explicitly restricting the situations in which the system is allowed to be used.

There is a further aspect that may make the CoSMoS approach more practical – as noted in Section 3.5, safety management and safety performance measurement are critical. Many aspects of system behaviour will only ever be discovered in real-world operation. If we can provide the initial confidence that a complex system is safe to operate, and have effective safety management in place, then we may be able to detect unexpected hazards before they occur and deal with them before they are able to cause an accident.

7 Conclusions

Safety engineering matters for many domains. In industries that are explicitly regulated (such as aviation or nuclear power) then conformance to explicit engineering standards is mandatory. Outside of those fields, it is still often the case that employers or product manufacturers have a duty of care towards their employees or users, and therefore need to ensure that the engineered systems they use are adequately safe. Increasingly, this requires system developers to present a structured safety case, incorporating argument and evidence, that a mandated level of safety has been achieved.

When producing a safety case, we need to present an argument that the system is acceptably safe, given the risk posed by the hazards present in the system. We do not need irrefutable *proof*, and we do not need evidence (however weak) of *total* safety. In particular, we don't need to make any claims at all about those aspects of the system that are not safety-critical.

Architecture is a powerful tool – we don't need ensure that every component of a system behaves safely, provided we can argue that the overall architecture system architecture will keep it in line. For example, it doesn't matter if a controller component proposes a hazardous act if we have a monitor component that can reliably veto that act. Complex systems, however, (along with micro-scale devices) prevent the use of many common architectural approaches and so present a major safety challenge. In particular, it is not at all clear how emergent properties could be bounded in this way.

In safety engineering, conservatism is a fact. It is present at every level, from working engineers through to governments (and then on to the general public). Complex systems are likely to face opposition simply because they are novel and little-understood, quite apart from the (genuine) technical difficulties they present. Resolving the technical challenges will help gain acceptable, but showing engineered complex systems working reliably in non-safety domains will be critical.

The real engineering of complex, safety-critical, micro-scale systems such as the AP system may be some way off, but we can start to work out what claims we will need to make, and how we can generate the evidence we need to make those claims. This way, when the technology becomes ready we may actually be able to use it.

Acknowledgements

The authors would like to thank Fiona Polack and Susan Stepney for illuminating discussions about the CoSMoS project, and for providing access to reference [21].

References

1. CoSMoS project team: Complex systems modelling and simulation infrastructure (2008) http://www.cosmos-research.org/caseforsupport.html, accessed 17 June 2008.
2. TUNA project team: TUNA: Final report (2008) http://www.cs.york.ac.uk/nature/tuna/outputs/finalreport.pdf, accessed 17 June 2008.
3. Health and Safety Executive: Health and safety at work etc act (1974)
4. Jones, M.: Development of an operational safety case within an existing safety management system. Master's thesis, University of York (2007)

5. RTCA and EUROCAE: DO-178B: Software considerations in airborne systems and equipment certification (1999)
6. UK Ministry of Defence: MOD interim defence standard 00-56 issue 4 — safety management requirements for defence systems (2007)
7. UK Ministry of Defence: Defence standard 00-56 issue 2 — safety management requirements for defence systems (1996)
8. Despotou, G., Kelly, T.: The need for flexible requirements in dependable systems. In: Proceedings of the 4th International Workshop on Requirements for High Assurance Systems (RHAS). (2005)
9. Pumfrey, D.J.: The Principled Design of Computer System Safety Analyses. Dphil, University of York (1999)
10. Kletz, T.: HAZOP and HAZAN: Identifying and Assessing Process Industry Hazards. 3rd edn. Institution of Chemical Engineers (1992)
11. Kelly, T.P.: Arguing Safety - A Systematic Approach to Managing Safety Cases. Phd thesis, University of York (1998)
12. Toulmin, S.: The Uses of Argument. Cambridge University Press (1958)
13. Motor Industry Software Reliability Association: MISRA-C:2004 — guidelines for the use of the C language in critical systems (2004)
14. Barnes, J.: High Integrity Software: The SPARK Approach to Safety and Security. Addison Wesley (2003)
15. Schneider, S., Cavalcanti, A., Treharne, H., Woodcock, J.: A layered behavioural model of platelets. In: ICECCS 2006, IEEE (2006)
16. Ritson, C.: A process orientated biological simulation using occam-. Technical report, University of Kent (2006)
17. Davey, P.: Medicine at a Glance. Blackwell Science (2002)
18. Ramrakha, P., Moore, K.: Oxford Handbook of Acute Medicine. 2rev edn. Oxford University Press (2004)
19. Liesner, R.J., Machin, S.J.: Clinical review ABC of clinical haematology: Platelet disorders. BMJ **314** (1997)
20. Kelly, T.P., Weaver, R.A.: The goal structuring notation — a safety argument notation. In: The Dependable Systems and Networks 2004 Workshop on Assurance Cases. (2004)
21. Polack, F.: Argumentation and the design of emergent systems. http://www-users.cs.york.ac.uk/∼fiona/PUBS/Arguments.pdf (2008)
22. Bloomfield, R.E., Littlewood, B., Wright, D.: Confidence: its role in dependability cases for risk assessment. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '07). (2007)
23. Littlewood, B., Wright, D.: The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN analysis of an idealised example. IEEE Transactions on Software Engineering (2007)
24. Weaver, R.A.: The Safety of Software - Constructing and Assuring Arguments. PhD thesis, University of York (2003)
25. Littlewood, B.: Limits to dependability assurance — a controversy revisited (or: A question of 'confidence') (2007)
26. Alexander, R.: Using Simulation for Systems of Systems Hazard Analysis. PhD thesis, University of York (2007)
27. MODAF Partners: MOD architectural framework executive summary (2005)
28. Padgham, L., Winnikoff, M.: Developing Intelligent Agent Systems: a Practical Guide. John Wiley & Sons (2004)

29. Padgham, L., Thangarajah, J., Winikoff, M.: Tool support for agent development using the prometheus methodology. In: Proceedings of the first international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005), Melbourne, Australia (2005)
30. Ammirato, F., Bieth, M., Chapman, O.J.V., Davies, L.M., Engl, G., Faidy, C., Seldis, T., Szabo, D., Trampus, P., Kang, K.S., Zdarek, J.: Improvement of in-service inspection in nuclear power plants. Technical report, International Atomic Energy Agency (2004)
31. Lam, D.N., Barber, K.S.: Comprehending agent software. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005). (2005)
32. Aviation Safety Network: Accident description 03 MAR 1974 (2007) http://aviation-safety.net/database/record.php?id=19740303-1, accessed 17 June 2008.
33. Muller, M.: Krebs cycle pictures and summary (2008) http://www.uic.edu/classes/bios/bios100/summer2003/krebsfull.htm, accessed 21 July 2008.