Deriving Safety Requirements for Autonomous Systems

Robert Alexander, Tim Kelly
Department of Computer Science, University of York
Heslington, York, YO10 5DD

Nicola Herbert
BAE Systems Military Air Solutions,
Warton Aerodrome, Preston, Lancashire, PR4 1AX

Abstract

In any safety engineering effort, deriving safety requirements is a key activity. Doing this for autonomous systems (AS) is challenging. However, several existing techniques can be pulled together to create a reasonable approach The risk of interaction between implemented requirements remains a concern, as does ambiguity about the appropriate boundary of the AS system. We believe these issues can be addressed by the development of advanced modelling and analysis techniques.

Keywords : safety, certification, requirements, analysis

Introduction

SEAS DTC project SER011 is concerned with the safety certification of autonomous systems (AS). We have previously produced reports and papers that surveyed the general situation ([1], [2]) and defined high-level certification objectives that are common across all AS ([3], [4]). Our current work, summarised in this paper, explains how those general objectives can be turned into specific safety requirements for a specific AS.

The primary aims of the work are (a) to identify a basic safety process that can be put into place on real projects and (b) to establish common ground for future discussions on AS safety.

As before, the work assumes that Def Stan 00-56 Issue 4 [5] is the standard to be certified against, as this is the primary safety standard for new MOD equipment acquisitions. Certification against 00-56 Issue 3 [6] may be possible, and there is little practical difference between these two issues.

The next section explains why safety requirements are necessary. This is followed by a section on defining the system to be analysed, and a description of the methods that can be applied to that system. We then discuss a salient issue (that of safety-critical interactions *between* requirements), and present a case study. This is followed by an outline of planned future work.

The Need for Safety Requirements

Put simply: safety requirements are requirements that, if met, will make the AS acceptably safe. They mandate actions that are needed for safety (for example, that a road-going UGV will stay in lane), and forbid things that will cause accidents (for example, an armed AV is not allowed to shoot at a target if a friendly is in the way).

It is *possible* that a safety-critical system could be developed without explicit safety requirements; certainly, non-critical systems are sometimes developed this way. It is unlikely in practice, because explicit safety requirements provide a key way to

maintain development team knowledge of what is important for safety. In typical practice, safety requirements are derived by a three-stage process:

1. **Hazard Identification (HI)** – all the *hazards* exhibited by the system are identified. A hazard is *"A physical situation or state of a system, often following from some initiating event, that may lead to an accident"* [5]. Part of HI is brainstorming the possible accidents that could occur, but this part is normally quite simple.

2. **Hazard Analysis (HA)** – possible *causes* of the system's hazards are explored and recorded. Essentially, this step identifies all processes, combinations of events, and sequences that can lead from a "normal" or "safe" state to an accident. Success in this step means that the safety engineer now *understands* how the system gives rise to an accident.

3. **Requirements Derivation** – once the set of hazards is known, and their causation is understood, engineers can derive safety requirements that either prevent the hazards occurring or mitigate the resulting accidents.

The Boundary of the System

When performing any analysis, we must define what constitutes the system to be analysed. We must define the *boundaries* of the system: what's in "the system"? What's relegated to "the environment"? We must also define groupings: what gets its own hazard analysis and top-level requirements? What just gets treated as mechanism, as part of something else? There are many boundary decisions you could justify, but we can make some firm recommendations.

The *ideal* is for analysis to take account of a wide context for the AS, including all its operators, peers and shared resources. A sophisticated high-LoA AS may have complex interactions with a wide range of peers. These peers include other AS, humans, and human-operated systems. Some of these peers will have the potential to cause hazards in the AS, and in turn the AS may be able to cause hazards for its peers. These ways to cause hazards need to be dealt with, so hazard analysis needs to include this wider context and requirements need to be derived both for the AS with respect to its peers (so it doesn't cause their hazards) and for the peers with respect to the AS (so they don't cause its hazards).

A *pragmatic* approach for many simpler AS (and simpler arrangements for use) is to treat "the system" as being the combination of operator(s), AS and any links between them. Hazards occur at the boundary of this system, and causes can be explained in terms of the parts within it (for example, a hazard at the boundary might be caused by loss of the communication link between AS and operator.). We refer to this combination as the "Combined Autonomous System" or CAS (after Hollnagel's "Joint Cognitive System" concept [7]). For clarity, the rest of this paper will assume the CAS as the object of analysis.

Methods for Analysis

In order to support engineers in the analysis of AS, we have identified a set of methods for each necessary aspect in the safety requirements process. The methods build on and complement each other in order give some confidence that all necessary safety requirements have been identified.

We expect that these methods, used together, will be basically adequate for much AS analysis. We are concerned, however, that many AS will need better methods. The potential for further work to improve on this is discussed in the section "Moving Forward", below.

Identifying Capabilities

A key first step is to identify the capabilities that the CAS needs to have in order to perform its mission and remain safe. It is very likely that its developers will already have identified some capabilities; it is also very likely that this will not be adequate for safety engineering. In some system design documentation, lower level capabilities may be implied rather than explicitly stated which may make the task of identifying capabilities difficult. Also, in some domains, standards may not exist, or if they do, may not only express capabilities in high-level terms.

There is a requirement for a robust and systematic method for identifying the capabilities required of a CAS. Task Analysis (see [8]) is proposed as a good methodology for identify CAS capabilities. Task analysis can be defined as the study of what an entity is required to do, in terms of actions and/or cognitive processes in order to achieve a given goal. *Hierarchical* task analysis, specifically, is a method of decomposing a high level capability down to its lowest levels in order to enumerate every capability required of a system.

The aim of using this technique is to produce a hierarchical decomposition of each of a CAS' high level capabilities, which is then repeatedly decomposed in more detail until the low level capabilities are at a level that can be usefully analysed for safety.

Hazard Identification and Analysis

The Energy Trace and Barrier Analysis (ETBA) technique is a preliminary hazard analysis technique based on energy models of accidents, where accidents are viewed as the result of an undesired release of energy from a system, which may lead to harm (see Leveson in [9]). The technique is based on the principle that if one can identify the sources of energy in a system, one can prevent an unwanted or uncontrolled release of that energy in a way that might cause harm, by using some form of barrier.

When performing preliminary hazard analysis, it is usual for an organisation to use checklists or previous hazard analyses, which encapsulate previous experience [10]. Checklists are useful as an aid to system developers in the preliminary identification of hazards, and checklists can be combined with other techniques such as ETBA. Obviously, the novelty of many AS makes checklists thin on the ground (and likely to be sparse when they do exist).

Scenario FFA is a method for doing hazard analysis over scenarios. An analyst works over a list of the events in a scenario, applying guide words to the events at each step. It is powerful because the context of any action is very salient to the analyst as he considers it (in contrast with FFA, which can be difficult because the method provokes analysts to consider functions "in general", without specific context.

Functional Failure Analysis (FFA) is a systematic predictive hazard analysis technique that is applied early in the development of a system design to help identify and refine safety related requirements [10]. The FFA technique involves the analysis of different failure modes of system functions, which must first be identified from an appropriate system design representation.

The Hazards and Operability Analysis (HAZOP) technique was originally designed for use in the chemical industry in the 1960s [9]. HAZOP is a systematic hazard analysis technique that allows a designer to examine all system flows and possible deviations from those flows using a set of guidewords (NO, MORE, LESS, REVERSE and so on). The aim of the technique is to identify all hazards associated with deviations from the

system's expected behaviour by examining deviations in system data flows.

A variant of HAZOP that can be applied to the hazard analysis of CAS was proposed by Hall May in [11]. Hall-May's Agent Behaviour Cycle (ABC), is a model of a system's behaviour based on Boyd's OODA loop (see [12]). The ABC identifies the stages in an entity's behaviour where failures might occur. Possible failure modes for a function are derived from the stages of the behavioural cycle of the ABC. A HAZOP-type deviation analysis can be performed on the functions of the system based on the types of failures that an agent can make.

The hazard identification and hazard analysis techniques presented here are intended to be complementary. First, ETBA and checklists provide a level of basic Hazard Identification from an initial system concept or set of system requirements.

The FFA technique provides a means to analyse the main functions of a CAS and the effects of deviations from those functions. The HAZOP technique provides a means to analyse the flows of data between components of a CAS and the effects of deviations from those flows. The Hall-May variant of HAZOP provides a means to analyse a system in terms of the kinds of error that an autonomous system might make when performing a task, such as making an error of perception when identifying a specific location.

Together these complementary hazard analysis techniques can be used to analyse hazards posed by a CAS from different viewpoints (with reference to functions, data and CAS behaviour). FFA derives hazards from known CAS functions, HAZOP derives hazards from expected data exchange with other systems, and Scenario FFA derives hazards from planned actions in expected scenarios. There is therefore a greater likelihood that all hazards and safety requirements can be identified in comparison to using only one of these techniques.

When performing hazard analysis, we must capture the resulting descriptions of hazard causation. A common form for capturing this is the Fault Tree. Fault Trees Analysis is a graphical technique which uses Boolean logic to describe the combinations of events and conditions that contribute to the occurrence of a hazard. Fault Trees are used in many fields including the aerospace, electronic and nuclear industries [9]. When reduced its "minimal cut-set" form, a fault tree shows the smallest sets of events which could lead to a hazard.

Requirements Derivation

Traditionally, most derivation of safety requirements has been informal. Engineers have looked at the hazard list and proposed requirements in an ad hoc fashion. The requirements are then analysed in a variety of ways (explicitly and implicitly) and re-worked as weaknesses are identified. It is a truism that changing requirements of any kind is expensive, because their scope for impact on the design, implementation and planned usage is large. A method that reduces the need for requirements change is always welcome.

To this end, we can have adapted the work of Hall-May on policy derivation [11]. The method was originally developed for creating safety policy for SoS. It is systematic, in that it provides a structured process for achieving complete requirements, and traceable, in that it maintains the rationale for each requirement.

The Hall-May method is based on existing methods for constructing safety arguments, such as those of Kelly [13]. It works by specifying top-level requirements and then progressively decomposing them until it reaches low-level requirements that can be

implemented. The resulting tree is an argument for the completeness and adequacy of the requirements. It is normally expressed in the Goal Structure Notation (GSN) that was developed for describing safety cases.

In our situation, we can get our top-level requirements from our identified hazards. For each hazard, we can state "Hazard X does not occur". We can then derive the levels below using the knowledge of hazard causation that we got from hazard analysis.

In Hall-May's work, he decomposed by assigning requirements to agents within a Systems of Systems. Compliance with those agent-level requirements was left to the individual agents (or their designers and operators). In our case, we need to go further – we need to assign requirements to the various component parts of the CAS. Such parts include the operator, advanced software components (such as planners and learning algorithms), software monitoring and safety functions, and mechanical components (such as actuators).

We may also, however, need to go 'upwards' from the CAS and impose requirements on the environment. These may be *SoS requirements* which are imposed on peers (e.g. "No manned aircraft shall approach within 1000m distance of the UAV, irrespective of other rules in force"), or they may be *use restrictions* on the AS (e.g. "Do not launch the UAV when average wind speed is above 20 mph").

In practice, a given high-level requirement may need to be decomposed across several parts of the CAS and its environment.

Interactions Between Requirements

There are many extant software safety standards, and safety standards that are concerned with software safety to some degree. Often, these standards state that the input to the software safety process is a set of safety requirements for the software function (stemming from a system-level hazard analysis), and the output is software that meets all those requirements. Effectively, the software safety requirements are handed-off from the system safety process to the software safety process. DO-178B [14] is the most relevant of such standards.

Although this approach may be adequate for simple software components, it is very fraught when dealing with highly complex, high-authority software. It is possible that the design and implementation decisions made to meet the individual requirements will interact – that the implemented software functions will interfere with each other. In other words, the overall behaviour of the system *emerges* from the interaction of all the implemented requirements. The resulting emergent behaviour may not be safe – it may introduce a new hazard, or provide a new way to achieve a known hazard.

This issue is relevant to any software-intensive system. We are concerned, however, that this problem will be worse for AS. In a manned aircraft, for example, there are many interacting systems that affect overall flight behaviour. Much of the time, the human pilot acts as a mediator between these systems.

By contrast, in an AS these diverse systems and functions will have to interact and cooperate without the human mediator. In a UAV the implemented capabilities for collision avoidance, staying out of restricted airspace, target monitoring, and respond to own health monitoring will all impact the same aspects of external system behaviour (in this case, gross UAV movement). In order to be safe, the effects of all these functions need to be coordinated.

It is therefore critically important to explore the interactions of software functions *as*

implemented; to explore the implications of the combined design and implementation choices you've made. If an AS has not been analysed in this way, then there is insufficient evidence that that AS is adequately safe.

00-56 Issue 4, as written, does not explicitly require this analysis. It might be possible to certify an AS without adequate modelling of software function interactions. However, it would be within the rights of a regulator (working to 00-56) to demand that this analysis be performed. In any case, in a court case after an accident, the developer would be open to a charge of inadequate engineering process.

One way to explore the interaction in a plausible context is to perform field tests. However, this is unlikely to be adequate for safety, and may not provide adequate evidence for certification (see our initial report [1] for more discussion on the value of evidence types). Indeed, in extreme cases this testing could *present* a severe safety risk. Therefore, some form of pre-field analysis will also be needed. The safety analysis team needs to model the algorithms, protocols and procedures implemented by the AS, and to model the missions and contexts that they will be used in.

This is an issue that we will return to as the project progresses. Some relevant approaches are discussed under "Moving Forward", below.

Case Study

To illustrate our combined method in action, we have applied it to one of the SEAS vignettes. Specifically, we chose Vignette 8A: "Air Attack - ISTAR Chain – SEAD". In this vignette, several different UAVs are used in order attack anti-aircraft platforms hiding in a built-up area. It showcases medium autonomy and high-risk operation.

We performed a task analysis on the case study, identifying a number of capabilities. From these (combined with a basic model of the interactions between the agents in the system) we performed a hazard identification and analysis. Some of the analysis results were captured in a Fault Tree.

Working from the identified hazards, we used the Hall-May method to derive some high-level safety requirements. Part of the Hall-May GSN structure is shown in Figures 1 and 2.

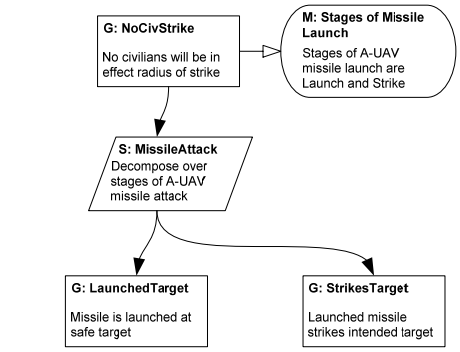

Figure 1 – Top Level Requirements Derivation

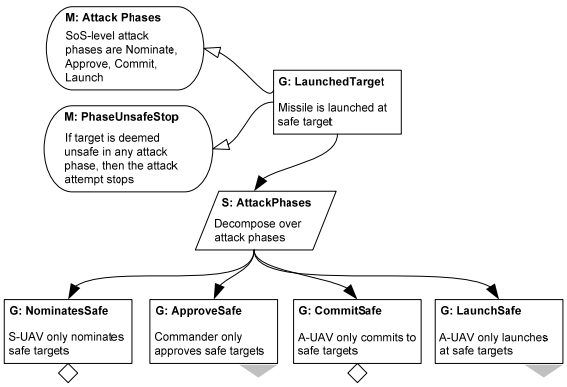

Figure 2 – Decomposition of 'LaunchedTarget' Requirement

In order to carry out this case study we had to make large numbers of assumptions. For example, just performing the Scenario FFA analysis on the scenario required us to make and capture fifteen explicit assumptions.

The full results of the case study, including a list of all assumptions made, can be found in our report [15].

Moving Forward

We will move forward from the work described in this paper in several ways.

First, our next work package concerns the wider safety lifecycle. The case study described above is typical of good early-stage safety work: it takes a Concept of Operations and explores its initial implications for safety. It uses many assumptions because detail is not available. This is not the end of safety engineering, of course – further work will be need throughout the lifecycle. As detail emerges (system architecture, software requirements, greater clarity of likely missions), its safety implications needs to be considered. Different techniques are needed (and useful) at different points.

Second, we will explore advanced analysis techniques to deal with the issues raised by autonomous systems. Relevant techniques include constructive simulation, model-checking and implementation in synthetic environments. A key advantage of all of these is that they integrate different aspects of the system description into one analysis. This is crucial for addressing the "requirement interaction" challenge discussed above.

At the time of writing, our SER011 collaborators in the advanced modelling and analysis strands (at QinetiQ and the University of Cranfield) are just getting started. We will work with them to incorporate their methods into our approach. Our primary project case study will show the use of these advanced techniques.

The challenge of justifying the validity of advanced models is ever-present. We will address this explicitly, and will present safety case patterns for arguing conclusions from these models.

Finally, we will apply our approach to further case studies. We hope to base these on the SEAS Vignettes, but in order to do this we will have to develop them well beyond their current level. The current descriptions are very conceptual, and as noted above even a rudimentary analysis requires a large number of assumptions. Safety engineering research is very difficult in the abstract – it becomes interesting only when applied to concrete cases.

In doing this, we will need support from the rest of the SEAS DTC. We are starting to look for partners and collaborators – other SEAS projects with systems and technologies that we can analyse. We hope to work with the SEAS demonstration programme in pursuit of this aim.

We are particularly keen to work with groups who are applying novel technologies in practical demonstration systems. We would like to work with you in order to define explicit safety requirements for a given use of your technology, and to explore some of the ways that you might meet those requirements. It may be that applying our work to your technology would provide grounds for a further funding bid.

Conclusions

We have developed an approach to deriving safety requirements for autonomous systems. The approach is widely applicable and can be carried out without extensive software or modelling support. We have a case study. In doing this, we have revealed some outstanding issues (interaction of requirements and assignment of a system boundary). These need to be addressed as the project moves forward.

References

[1] R. Alexander, M. Hall-May, T. P. Kelly, "Certification of Autonomous Systems," SEAS DTC SEAS/TR/2006/1, (2007).

[2] R. D. Alexander, M. Hall-May, T. P. Kelly, "Certification of Autonomous Systems," in *Proceedings Of The 2nd SEAS DTC Technical Conference*, Edinburgh, (2007).

[3] R. Alexander, N. Herbert, T. Kelly, "Certification Objectives for Autonomous Systems," SEAS DTC SEAS/TR/2008/1 (2008).

[4] R. Alexander, N. Herbert, T. Kelly, "Structuring Safety Cases for Autonomous Systems," in *Proceedings Of the 3rd IET System Safety Conference*, (2008).

[5] "MoD Interim Defence Standard 00-56 Issue 4 - Safety Management Requirements for Defence Systems," Ministry of Defence, (2007).

[6] "MoD Interim Defence Standard 00-56 Issue 3 - Safety Management Requirements for Defence Systems," UK Ministry of Defence, (2004).

[7] E. Hollnagel, D. D. Woods, *Joint Cognitive Systems: Foundations of Cognitive Systems Engineering*: CRC Press, (2005).

[8] B. Kirwan, L. K. Ainsworth, *A Guide to Task Analysis*: CRC Press, (1992).

[9] N. G. Leveson, *Safeware: system safety and computers*: ACM Press New York, NY, USA, (1995).

[10] D. J. Pumfrey, "The Principled Design of Computer System Safety Analyses," DPhil Thesis, University of York, (1999).

[11] M. Hall-May, "Ensuring Safety of Systems of Systems—A Policy-based Approach," PhD Thesis, University of York, September 2007, (2007).

[12] J. R. Boyd, "A discourse on winning and losing," Air University Library, Maxwell AFB, Alabama, USA Tech. Rep. MU43947, (1987).

[13] T. P. Kelly, "Arguing Safety - A Systematic Approach to Managing Safety Cases," PhD Thesis, University of York, (1998).

[14] "DO-178B: Software Considerations in Airborne Systems and Equipment Certification," (1999).

[15] R. Alexander, N. Herbert, T. Kelly, "Deriving Safety Requirements for Autonomous Systems," SEAS DTC SEAS/TR/2009/1, (2009).

Acknowledgements

The work reported in this paper was funded by the Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre established by the UK Ministry of Defence. We would like to thank Andrew Miller (BAE Systems), and Richard Hawkins (University of York) for their help and support in this work.