Safety Assurance for Operating Procedures – a formal methods approach

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

p.lindsay@uq.edu.au

Abstract

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

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

Keywords: ATM, formal methods, operating procedures.

1 Introduction

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

1.1 ATM operating procedures

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

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

1.2 Use of formal methods

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

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

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

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

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

2 Background: Trajectory Based Conflict Detection and Resolution

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

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

2.1 Conflicts and separation standards

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

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

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

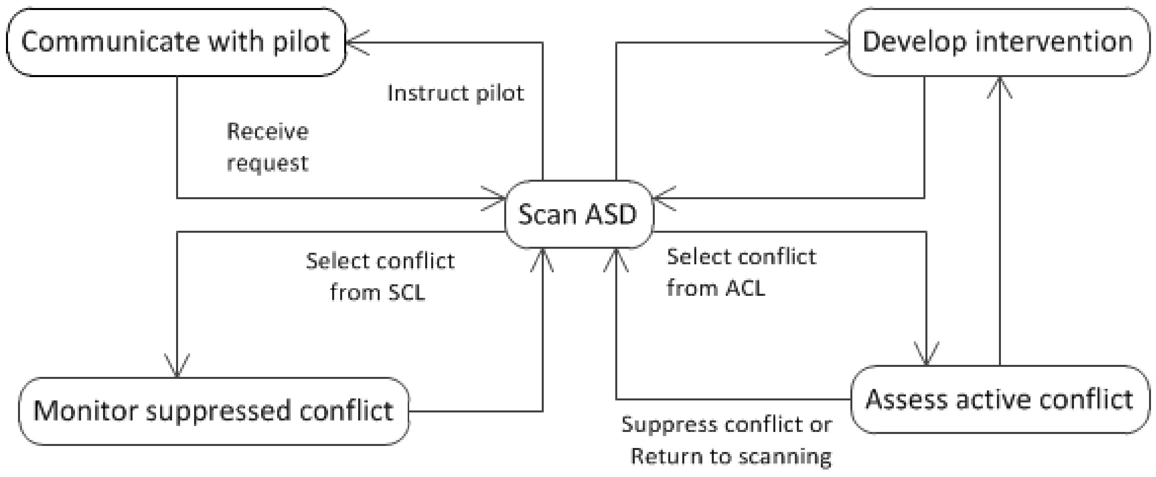

Figure 1. TBCD operator activities

2.2 TBCD operational concept

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

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

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

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

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

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

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

2.3 Standard Operational Procedures (SOPs)

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

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

SCL. (The ‘needs assessment’ flag gets deleted automatically.)

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

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

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

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

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

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

3 Hazard Analysis

3.1 Hazards

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

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

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

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

3.2 Hazard formalisation

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

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

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

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

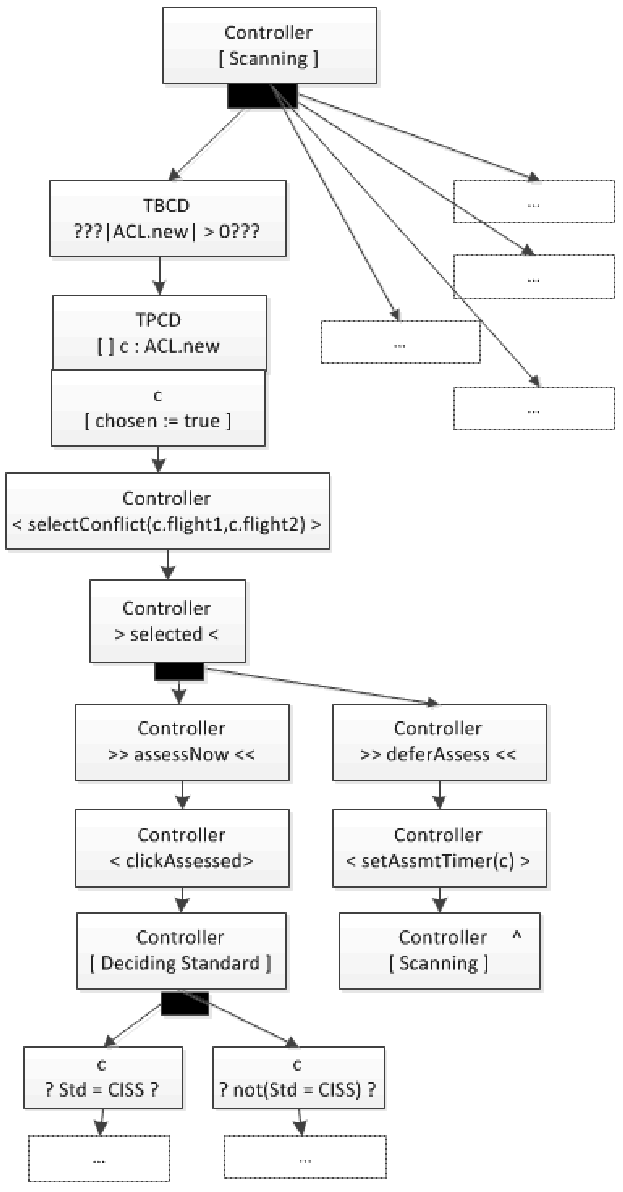

Figure 2. Excerpt from the BT model

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

3.3 Operator failure modes

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

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

Examples of the errors generated by this process are:

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

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

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

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

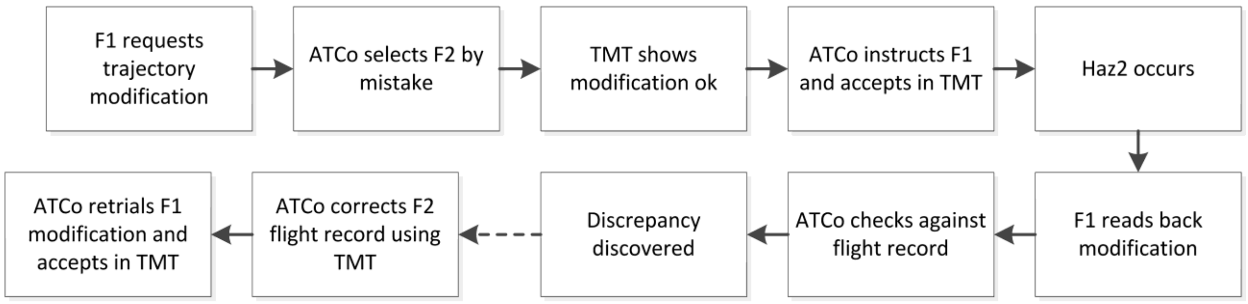
F1 requests trajectory modification
ATCo selects F2 by mistake
TMT shows modification ok
ATCo instructs F1 and accepts in TMT
Haz2 occurs
F1 reads back modification
ATCo checks against flight record
Discrepancy discovered
ATCo corrects F2 flight record using TMT
ATCo retrials F1 modification and accepts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

Figure 3. Accident sequence leading to Haz2

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

3.4 Model checking

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

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

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

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

4 Design Improvement

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

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

5 Discussion

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

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

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

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

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

6 Related work

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

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

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

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

7 Conclusion

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

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

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

8 References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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