Pilot interaction with artificial agents

Em L. Tonkin

MSc. HCC
Department of Computer Science
Faculty of Science
The University of Bath

June 2002

Abstract

Working from a runway overrun accident report, several causes are identified, principally lack of situational awareness, insufficient training, and communication breakdown. Possible solutions are discussed, particularly combination of data retrieval/mining techniquess with a number of agents including a context definition system and a simulation/watchdog system, outlined in Prolog, based around the blackboard model. A software architecture permitting this system to coexist with the pre-existing aircraft avionics is outlined, and the implications discussed in terms of human-computer collaboration, social issues, and team structure.

Contents

With thanks to:

Greg (who 0wn3 LaTeX)

Steve, Vai

and DALnet's resident psychologist,

Deadbert,

who sent me the Booth book.

The world can be divided into two types of people;

those who divide the world into two types of people, and those who don't.

1 Introduction

Interface design can make a great deal of difference in safety-critical applications. This effect is quite clear in the case initially chosen for study —the crash of a Boeing 747 aircraft in Bangkok, Thailand, which took place on the 23rd September 1999. Analysis of this crash has shown a number of contributing factors. These, once identified, can be used to illustrate problems with the interface used on board the plane.

These problems are discussed and methods by which they may be ameliorated are identified and outlined . However, there remain a number of issues —for example, how and when automatic systems may volunteer information, and determination of the importance of that information to the current situation as perceived by the human pilot.

It is visibly true, with reference to the case study, that the existing interfaces are somewhat inadequate, providing little feedback. Information is provided in an inaccessible form and automatic functions are often hidden.

It is therefore suggested that interface design in systems of a certain nature (for example, the autopilot system of an aircraft) could be redesigned in such a way as interaction is more 'natural', and effective for the users. This could be by changing (or extending) modalities of the interface, or by incorporating systems such as intelligent agents for data analysis so that raw telemetric data may be translated into a few relevant conclusions for presentation to the user.

2 Introduction to the Bangkok incident

Information on this case has been taken from the Australian Transport Safety Bureau's investigation on the subject.

On 23 September 1999, at about 10:47 PM, a 747 landing at Bangkok International Airport overran the runway, sustaining substantial damage. No serious injuries were reported as a result of the crash. The flight carried 391 passengers, 16 cabin crew, and three flight crew members. The aircraft landed 'long', which is to say, mid-way down the runway, and due to the heavy rain at the time of the landing also aquaplaned, meaning that the braking was ineffective and as a consequence the aeroplane overran.

3 Detailed introduction

The flight crew members included pilot and first and second copilot.

The crew was informed by air traffic control during the runway approach that the weather conditions were bad — a thunderstorm and heavy rain at the airport — and that visibility was 4km or greater. However, the visibility degraded as they approached the runway, 1,500m at 22:40 and runway visual range for the runway used, 21R, as 750m. The crew was not informed of this information nor the fact that another aircraft had 'gone around' from final approach, which is to say, aborted their landing procedure in favour of making another approach. The crew was informed at 22:44 that the runway was wet but that a preceding aircraft reported that braking action was 'good'.

The crew noted the adverse weather conditions late in the final approach, at which time the angle of approach of the aircraft deviated. The deviation was small enough to be within limits suggested by the airline, Quantas. When the aircraft was approximately 10ft above the runway, the captain instructed the first officer to go around.

The first officer therefore advanced the engine thrust levers, as the aircraft touched down 636 metres beyond the ideal touchdown point on the runway. The captain then changed his mind, cancelling the go-around by retarding the thrust levers. He did not announce this. The crew were therefore somewhat confused as to the captain's intention.

During landing, the flight crew used settings of flaps 25 and idle reverse as the approach and landing configuration, based on normal company practice. This was inappropriate, due to the conditions and their exact position on the runway —a better approach existed (flaps 30, full reverse thrust) that would have permitted a safe landing according to the documentation that Boeing provided on the aircraft. This error was compounded by later events.

The aircraft continued, without braking, down the runway due to a number of failures:

The captain's use of the engine's thrust levers to attempt a 'go-around' was not the optimum method. Instead, a system called TO/GA, after Take Off/Go Around, existed to do a similar task. The TO/GA system was, however, not transparent in its functionality. The captain had the erroneous idea that using the engine thrust levers would cause the go-around to execute more quickly.

When the captain aborted the go-around, he did not completely retard the thrust levers. This caused the automatic systems that control the specifics of braking to make the assumption that the intention was not to land, and therefore to disengage the autobraking system. The crew did not notice that the auto-brake system had disarmed. Instead, the first officer commenced manual braking, since 'the aircraft did not appear to be slowing down'.

The crew did not select reverse thrust (redirecting the engine jet in the direction of motion in order to decelerate). Therefore it was not until the aircraft reached halfway down the runway that the aircraft's speed began to diminish.

A significant factor during the landing roll was dynamic aquaplaning, an effect that occurs at these velocities when 3mm or above of water is present on the runway — as an effect of excess water covering the runway, the tyres of the aeroplane lost contact with the runway, instead sliding along the water's surface in an almost frictionless manner. This reduces the effectiveness of the wheelbrakes to about a third of that on a dry runway. Therefore, only the use of reverse thrust could have slowed the aeroplane in time to avoid overrunning the end of the runway.

In practice, the aircraft overran the stopway at the end of the runway at 88 knots (each knot is 1.852 kilometers/hour, which makes the speed of the plane at the end of the runway approximately 162km/h). It finally stopped 220m later at or by an airport perimeter road.

During the overrun, the aircraft was damaged, including numerous cabin fittings that were

dislodged.

During the landing roll, the cabin steward commenced the 'Welcome to Bangkok' passenger address announcement, stopping part-way through when he sensed that the landing was not normal. Some of the cabin crew also sensed this and adopted the 'brace for impact' position. However, it was not until after the first impact, after the overrun began, that any members of the cabin crew informed the passengers of the danger (by calling "heads down, stay down" or "brace brace brace"). No previous warning was given to the passengers, leading to the result that only forty-five percent of all passengers reported that they braced during the impact sequence. Many 'did nothing', whilst seventeen percent 'just held on tight'. At least one passenger had undone their seatbelt when the impact occurred, believing it to be a normal landing.

4 Analysis of the Bangkok crash

To summarise the points of failure:

- The flight crew did not react correctly to the conditions (did not use an adequate risk management strategy for approach/landing);
- The first officer did not fly the aircraft accurately during the final approach;
- The captain cancelled the go-around decision by retarding the thrust levers;
  - The silent reversal of the go-around decision caused confusion;
  - It also resulted in excess thrust after touchdown, delay in spoiler deployment;
  - Finally, it caused cancellation of the autobrakes;
- The flight crew did not select idle reverse thrust, or notice that it had been cancelled. Nor did they select full reverse thrust, due to a lack of familiarity with the system;
- The runway surface was affected by water —a physical issue;
- Also, the crew was not aware of the bad visibility, nor of the fact that a preceding aircraft had gone around;
- They did not notice that the auto-brakes had disarmed due to the manipulation of the engine thrust levers;

5 Responses to these difficulties

It is clear from the above description that certain factors contributing to the accident can be put down to lack of information. However, much of this information was available, at least in the sense that one of the entities in the system was aware of it.

Captain	Intentions (to take off/land)
ILS	Deviation from correct approach to runway
Previous flight approaching runway	Visibility near the runway
Control tower	Degraded visibility near the runway
Control tower	Velocity and direction of plane (radar)
Autobraking system	Wheel rev/runway conditions
Autobraking system	Actual status of brakes (engaged/disengaged)
Autopilot systems generally	System's conception of intentions (takeoff, etc)
Flight crew	The risk of overrunning the runway
Boeing engineers	The actual capabilities/suggested procedures for the plane
Cabin stewards	Procedures for interaction with passengers

It seems that the flaws of interaction in this case were not only between the plane's autobraking/autopilot computer systems but also between the various human entities involved. For example, the cabin crew were not explicitly informed of the danger, although some of them took their cue implicitly from the unusual 'feel' of the landing and the fact that the captain did not complete the usual 'Welcome to Bangkok" address. Equally, the control tower did not mention the additional facts of which they were in possession.

Furthermore, much of the information that would have enabled the flight crew to choose an optimal solution to the conditions was unavailable to them in the timeframe of the landing. The case study notes that much of the information initially compiled by Boeing engineers was not distributed widely, and even had it been, the probability is that the information would not have been presented in sufficiently simple a manner to be useful

in a restricted/stressful situation. The suggestion made by the case study is that pilots should be given further training in the areas useful (for example, procedures for landing on contaminated runways). However, the question of whether it is practical or cost-effective to provide detailed training, sufficiently current for it to be useful in a stresssful situation, of every conceivable situation is an open one.

Some of the difficulties that contributed to this interaction can be solved by making alterations to standard operating procedure for human-human collaboration, and possibly by providing computer support for elements of that interaction. However, much of the difficulty lies with the onboard systems' treatment of the available telemetric data. The third issue is that of the human pilots' interaction with the autopilot/autobraking systems. The problem, for example, that caused the brakes to disengage due to the autobraking system's implicit assumption that since one of the engines remained on, was difficult to diagnose without relatively advanced understanding of the exact working of the automatic systems. The pilots did not have this knowledge —since in most occasions the autobraking system's assumption that 'thrust on == brakes off' was correct, the rules by which it operated generally remained unnoticed.

The autobraking system's implicit assumption as to the intentions of the pilot is a way of establishing the context within which the system is currently operating. However, it has been previously noted in discussions of context, such as that of Svanæs (2001) that allowing a system to interpret the user's action based on assumptions about the user's intention requires that the user's intention be deduced from previous actions and properties of the surrounding environment, which he further notes that, in most cases, cannot be done. Svanæs suggests that enabling a user to resolve ambiguities directly is easier.

However, in a system as complex as that of an aircraft, this is perhaps not quite such a simple proposition. The software designed to control the functions of large aircraft must by necessity provide at least one layer of abstraction between the hardware and the user. The functionality important to the user is not, in most cases, 'raw' functionality, and the

data that the user finds helpful to complete their goals is not raw — furthermore, the differences between individual aircraft are likely to make a certain amount of standardisation extremely desirable. Therefore, it seems very difficult to escape the conclusion that the system will have to act according to a given set of circumstances. The question becomes how the system should classify possible states, how it should react to user input and how that input should be provided.

The autopilot breaks the 'rules' by not having a formal interface of any kind and no clear query method. This is because it is supposed to remain largely unperceived (eg. a collection of functions rather than a separate entity in its own right).

The effect of the current interface between the flight crew and the automatic systems is that the pilots have little, if any, awareness that a computer system with the ability to take control of systems such as the brakes exists. This works well provided that the system makes no errors; otherwise, rectifying such errors becomes very difficult.

There is also a requirement for additional (non-interactive) volunteering of information. In this case study, this would have proven to be useful, for example, for the cabin crew, who were given no formal warning of the probability of crashing and were therefore unable to warn the passengers/take appropriate action. Equally, the control tower/airport staff may have benefitted from the provision of further information on the results of the landing in order to mobilise the emergency services more quickly.

6 Data retrieval and analysis

> "Especially in novel situations, team members must communicate to respond to environmental cues, explain to each other why previous strategies do not work in the novel situation, jointly determine new strategies, and predict future states" - Orasanu, 1990 [12]

The question arises; just how much information should be given to the pilots? This question is intimately involved with others; how much control should be given to the automatic systems, how is that data collated, and in what form should that data be presented?

A brief discussion of the data types most likely to be of use to the pilots would therefore be useful —this should be considered preliminary, since this in itself would be, it seems, a fit subject for further study.

- Purely factual information, in 'raw' form
- Factual information, following analysis — for example, in the case of aquaplaning, a number of variables combine to produce the conclusion of which the individual datapoints each portray a single element (the revolution speeds of the front vs. the back wheels and the speed of the plane).
- Predictions — for example, the prediction that a given strategy during landing will result in overrunning the runway. These cannot be considered canonical; whilst physics provides an excellent capability for analysis of certain situations, mostly somewhat idealised, effects such as thermal currents, changes in windspeed or direction and weather conditions could significantly alter the result. Therefore, such calculations provide information as to probabilities rather than certainties. They provide constraints and extremes.
- Volunteered data — it would seem inadequate to give only sufficient information for the user to be aware that their given strategy is ineffective. In the case study,

it is noted that Boeing initially provided tables of information designed to give the optimum strategies for implementation in each situation. This situation, that of a contaminated runway (contaminated, that is, with sufficient water to produce aquaplaning) combined with short stopping distance, is covered by these tables. Indeed, most situations are. These tables, however, are seldom given to pilots, and in any case would be difficult to apply directly. Thus, this and other information, chosen for the situation, could usefully be volunteered.

- Requested data — pilots may wish to request data. In the case of, for example, an equipment failure, they may wish to request additional data on the problem and its severity.

7 Possible architectures for a context-aware system

A number of architectures have previously been suggested for the management of context information and the design of context-aware software. Note, however, that few of the designers of these architectures have agreed upon a common understanding of the term "context", and therefore each is effectively designed to hold a different subset of information or access it in different ways. In this report, the word *context* is taken to mean both physical location and state, intention, and social conduct (or as Agre puts it, architecture, practices and institutions).

Some existing architectures attempt only to quantify a small amount of physical information, such as the Microsoft Research example[58] in which, although their definition of *context* is "situational information relevant to the interaction between a user and an application", the contextual information is primarily to do with location. In itself, this paper provides little useful information on the subject of architecture, preferring to focus on the results. However, their suggestion that making use of physical objects as UI devices is interesting. Ishii [38] suggested that physical objects may form part of a user interface, and furthermore suggested the catchy term *phicons* for objects that may be manipulated to direct a system, display some activity to represent something, or be placed in a certain way to associate with a given meaning.

Again, elements of the interface in the cockpit already make use of the 'phicon', although they are not named as such.

Hong and Landay[36] have suggested a design based on service infrastructures, promoting the use of network-accessible middleware infrastructures for context-aware computing. Their conclusion is that a service infrastructure for context awareness is of benefit due to its independence from hardware, operating system and programming language, simplicity of maintenance and evolution, and the ability to share sensors, processing power, data and services.

However, they also point out a number of challenges that are relevant to this report;

- Defining standard data formats and protocols;
- Building basic infrastructure services;
  - Automatic path creation;
  - Proximity-based discovery;
  - Apportioning Responsibilities;
  - Scoping and Access of Sensor and Context Data;

Dey[29] propose a widget model, where interaction is described in terms of callbacks and messages. Dynamic data is available from distributed sources. However, this model is far from standard designs or standards. Dey has defined context as the user's physical, social, emotional or informational state. The model described here is, however, restricted to places, people, and things (identity, location, status and time being the essential categories of context information). One feels that much of this work is most directly applicable to everyday situations, rather than a highly specialised system such as supporting a flight crew.

Engelmore and Morgan[32] propose a blackboard model. In this model, each process in ownership of a piece of information posts it to a shared blackboard — notice board — which is available to other processes. If information is not yet available, a process may request it, in which case they will be sent the information when it becomes available. However, this requires a central server to be in place in order for the information to be stored.
Winograd[74] describes these three models and provides a comparison between them, describing the criteria upon which to choose between them as:

- Efficiency (bandwidth, latency, etc)
- Configurability (the difficulty of configuring a system involving multiple processes and devices — some of which may not be under the designer's control)
- Robustness, the effect of component breakdown, and error handling mechanisms

- Simplicity, the ease of use of the system for the designer

Winograd also notes that the widget model is designed to be tightly coupled together, closely controlled and efficient, but as a downside requires complex configuration and is less than robust. The service model is a great deal more robust, but efficiency and control could be a problem — note that whilst Winograd states that it puts much less emphasis on efficiency and tight control, this is not necessarily the case. Should one decide that a given latency is required, for example, this may be achieved in the same way as it would have been achieved on a system that does not depend on a network infrastructure. Similar connections are often made (for example, private modem networks using any protocol to connect to a HTTP-using gateway, which for other users interacting via that gateway is a standard connection, but for that user is whatever system they connected with). At worst, a very specialised network system simply becomes that which would have been designed if the network was not available (an example of this is Internet2, high speed internet connections between universities and capable of fantastically high bandwidth/latency).

The blackboard model is noted to pay a price in communication efficiency, since the components are very loosely coupled together, and each communication requires two hops and uses a general message structure that is not optimised for any particular kind of data or interaction protocol. However, ease of configurability and robustness is a compensatory factor. Again, note that the message structure may surely be optimised, particularly for a case such as the one in this report where the topic in question is rather specialised and the network is not intended to be open to the world in general.

Winograd's comment that "There is no panacea: Each solution will be suited to some environments and not to others" seems like a good place to end this discussion.

8 Intelligent agents, information gathering, and analysis

> Machines should work. People should think.
>
> – IBM's Pollyanna principle.

The information required comes from a variety of sources: air traffic control, telemetry, the pilots, background knowledge on the destination, Boeing's reference material, training, and other automated systems such as the ILS... One can already see the knowledge retrieval systems collectively referred to as 'telemetry' taking control of the collection and analysis of data from other sources and producing factual information by analysing the raw data. With this functionality in mind, it is a short step to imagine the telemetry system as some sort of intelligent agent, that proactively measures variables, analyses them, and makes them available for use in the interface.

It is unfortunate that the phrase 'intelligent agent' has received so much recent attention, since the effect is that there are now many definitions of the term[71][69], many of which are so overbroad as to be largely devoid of meaning. For the purposes of this paper, Parunak's 'lowest common denominator' description will be used — an agent is 'a proactive object', with the data and code encapsulation of a software object, its own thread of control (making it an active object), and the ability to execute autonomously rather than being invoked externally(making it proactive). These agents interact with each other by way of their shared environment, for example, a network.

Holland[35] argues that every agent that operates in a changing world must model that world internally, whilst Parunak[71] notes that the sophistication of the knowledge representation and reasoning used may vary. In this case, the prime purpose of the telemetry system is to provide as accurate a model as possible for the current situation. The telemetry system's very purpose is to explicitly model aspects of the world. The telemetry system operates as part of a system made up of the surrounding environment and other agents, but exclusively concerns itself with the present, reporting and analysing current information.

However, the telemetry system stores information that could be used to build up models of the probable future. This could perhaps be requested and used by other agents that do not, as yet, have a real-world analogue in the cockpit; for example, an agent designed for analysis of the effects of current strategy would make use of telemetry in order to reason out a probabilistic response and return it to the originator of the request. Equally, an agent designed to volunteer information to the pilots would perhaps work on the basis of information gleaned from telemetry.

One major shortcoming of this approach in the present-day cockpit is the method by which information is transferred from the control towers at the airport to the pilots, which is, on the whole, by spoken-word radio transmission. Analogue radio does not provide a good data interchange format, and there is no suggestion that this data is transmitted in any form more palatable for software products. For this approach to be used to its full capabilities, such information should be available on agent request. How this is most plausibly done —whether the agent should request the information from an external source or the information is volunteered to the agent — is a quite separate issue.

One of the greatest strengths of agent architectures is their distributed nature[21]. Since each exists essentially as a separate process from the others, perhaps on a separate system entirely, there are few situations in which the failure of a single agent can cause others to fail (in the sense of crashing — they may well fail in the sense of being unable to accurately complete the task). Furthermore, it suggests that agents, as with many modern systems such as decentralised peer-to-peer applications, are capable of actively seeking out other sources of information where they are available. Such an ability is of some use in this case.

For example, the fact that a previous flight that had attempted to land on the runway some five minutes previously could have been of some use to the decision-making process. This fact was known to a number of people, for example the airport controller who spoke with the flight crew of that plane. However, it was not common knowledge in the sense

that it was not shared between every entity involved, and this, in the current system, is broadly speaking both acceptable and inevitable. Many planes land in any given time period, and to be required to inform every flight crew of the status of all of the more recent previous flights is an unacceptable load on both crew and controller. However, if this information were to be stored electronically by some system, such as a database, the fact could be retrieved and taken into account by a 'watchdog' agent. This multiplication of the number of entities involved causes some confusion; each plane gains a team of software programs each working with no direct human control, and very probably, the ground-based control systems suffer the same fate. Data is transmitted as and when software demands it.

This also leads to a further difficulty — accountability. Software, as a rule, leaves the difficult decisions to human users. The issue of accountability is, therefore, solely on the conscience of the user. However, the bandwidth of communication between the agents in such an architecture as has been described above is potentially rather high. Much of this information cannot be presented to the human pilots for review, either because it is in forms meaningless to a human, which may of course easily be changed by filtering/analysis, or because there is simply too much of it. Information must be condensed and perhaps simplified before it may be transmitted to a human. If the result of this is that a pilot makes a decision that he or she would not have made if all of the information had been available, or perhaps a single piece of vital information, does this make the system unacceptable?

Additionally, of course, the authenticity of a data source must be certain. As human pilots prefer to take information from trusted sources, so should the agents. The networks on which they act may not themselves be trustworthy[49]. The establishment of secure VPN-style connections is plausible. However, assuming that a system is in widespread use, the assumption that every trusted user on that network is worthy of the trust is naive; better to learn a lesson from the design of modern public key cryptographic systems such as PGP[1] and its VPN descendant, PGPnet. Message hiding, debatably, is not the issue

[1] www.pgp.net

so much as message authenticity. Weather data does not need to be hidden, but it must absolutely be secure from falsification, just as with other systems such as the ILS[2] beacons. False information could lead a system such as this one into a situation where any information it provides to the pilots is likely to be harmful. However, extremely similar technical issues have been dealt with and documented previously.

The heuristics that should be used to classify the importance of a given piece of information would require a great deal of study to define correctly. In fact, the probability is that there are no correct answers. A heuristic, like any general function intended to describe a more complex process, can only approach a solution to a reasonable accuracy. Furthermore, a heuristic incorrectly defined (for example, intended to describe a system in which the same inputs lead to two extremely different outputs), even if designed in order to alter itself from positive or negative feedback, may well be incapable of describing the span of solutions and thus the process of 'learning' has no positive effect. If, however, such rules are carefully defined and observed (particularly if designed to learn from feedback, as has been suggested[33]).

The system does not have to be designed in as anarchistic a way as the agent architecture would permit. Architectures like CORBA[72] (Common Object Request Broker Architecture, one of the Object Management Group's standards) could provide a helpful reference, providing a proven practical solution to many of the difficulties of running services across heterogeneous networks.

[2]http://www.navfltsm.addr.com/ils.htm

9 Prediction

So far, the use of agents to gather information and perform basic (factual) analysis, such as application of equations to extract solutions that are essentially deterministic and subject to no significant requirement of qualification, has been discussed. However, this covers only two of the various types of data noted above. The others also, from the example of this crash, stand as demonstrably useful qualities.

The capability of prediction of events is not, physically speaking, a particularly difficult task. The important variables are likely to be: the weight of the plane, its velocity, its pitch, roll and yaw, the velocity and direction vectors of the wind, moisture/precipitation,, the setup of the wings (curvature) that affects lift, and the altitude. Effects such as the 'ground effect' come into play at certain altitudes. It is necessary to note that mathematical simulation of the effects of a given set of variables (pressure, etc) in aerodynamics is quite accurate and relies almost entirely on accurate measurement of pressure. However, pressure is not a constant, and whilst it may be measured with good accuracy at any one instant, the accuracy of that mathematical prediction depends on the accuracy by which local pressures are forecast. Thus, a prediction is at best well-informed guesswork.

Fully-fledged computational aerodynamic prediction is a complex subject, and no attempt will be made to describe it here. A good introduction can be found at the Sydney University Aerospace Department pages[3]. Suffice it to say that computational aerodynamic prediction cannot — with present-day computer systems, at least, on anything less than an inconveniently heavy supercomputer — be executed in real time. A more reasonable approach may be to complete predictions based on the work/measurements done by Boeing. For example, Boeing have published relatively extensive tables, including correction factors for the relevant variables, permitting the stopping distance on a runway in any given condition to be known. Noting that some of these variables are themselves difficult to predict (for example, a runway may become contaminated very quickly), this, together

[3] `http://www.aeromech.usyd.edu.au/aero/contents.html`. An alternative site with similar coverage may be found at `http://www.aoe.vt.edu/aoe/faculty/Mason_f/CAtxtTop.html`

with a table of results from a simulation of descent, could permit a rough estimate of the viability of a given landing approach pattern. It could also, rather more usefully perhaps given the rough nature of such predictions, provide an estimate of the result of any alteration of that approach.

10 Volunteered data

The ability to process and analyse information, or the ability to predict, are not in themselves very useful —particularly the second. Rather than providing only the ability to give a probabilistic response to the viability of a given set of parameters, the ability to volunteer optimal settings for a given set of conditions could be provided. This functionality could, if implemented, permit the agent responsible for 'watchdog duty' to notify the pilot of the problem (landing too far down the runway, or braking ineffectually due to weather conditions), as well as the proposed solution, calculated by choosing the nearest match to the current situation from the Boeing tables. In essence, this functionality should be used fairly seldom. The system should preferably not control the plane entirely, unless there is an extremely pressing reason to do so.

The role of the system is set as advisor, rather than active crewmember, principally due to the limitations of systems that attempt to make extensive use of context. It is frequently noted that systems of this kind should be designed in order to give (and accept) frequent feedback in order to ensure that the users of the system are provided with sufficient information to understand its actions/suggestions.

11 Managing contexts

The case study shows a good example of a breakdown in understanding of context between the automatic system and the human pilots — the pilot's decision to go around, closely followed by the captain's unilateral and unstated decision to land, was sufficient to confuse the software into failing to react correctly. Whereas, with a clear understanding of the context, the most appropriate response would have been to apply the brakes and reverse thrust fully and turn off the motor that the captain had left activated, this is less than clear for a system in which intent/context is taken as implicit in the user's actions.

For this reason, the original system was sensitive to failures of communication. This implies that a solution is required; either the provision of systems that provide explicit feedback, or perhaps a method of examining the assumptions made by reference to the actions/telemetry. The latter involves the adoption of some sort of logic in order to identify scenarios such as the above as 'impossible', in that they are situations which could not be intended.

This implies that the system should be capable of examining each element of input and weighting the effect of that input on its conclusion. For example :

- A The plane is currently on the ground
- B The plane has just completed a descent (early stage of landing)
- C The pilots are attempting to engage the brakes
- D The configuration of the flaps implies landing (flaps 25)
- One of the engines is producing forward thrust (more generally, the condition for landing is E *All forward thrust is off*)
- F Main wheels are engaged with the ground

Analysis of the facts above could be completed using logic.

p: The plane is landing ($\overline{p}$: the plane is not landing).

q: The plane is in the early stages of takeoff.

r: The plane is stopped (landed).

The aeroplane's systems currently react as follows; its conclusion on the plane's status (landing/not landing) is $\overline{E} \rightarrow \overline{p}$, which may be restated in English as "All forward thrust is not off, which implies that the plane is not landing". The contrapositive; if the plane is landing, then its forward thrust will be off ($p \rightarrow E$). A number of similar statements can however be made:

- The plane can be in one of four general conditions; p, q and r are essentially mutually exclusive. Thus, either the plane is in p where q and r are false, q where p and r are false, r where p and q are false, or in none of the above states (p, q and r are all false).
- If the plane is on the ground, then it is either in the late stages of landing — $A \rightarrow p$, (or halted $A \rightarrow r$) or in the early stages of taking off — $A \rightarrow q$
- The plane is landing if the plane is on the ground and the plane has *just* completed a descent (barring very strange geography).
- Thus, $(A \cap B \rightarrow p)$. More generally $(A \cap B) \rightarrow (p \cup r)$, somewhat as before (*just* is assumed to be small).
- The Take Off/Go Around system that is capable of aborting landing at the last minute influences this by permitting a plane on the ground to take off again at very short notice; in 'emergency situations', the fact that the plane is on the ground and has just completed a descent does not preclude taking off: $(A \cap B) \rightarrow (p \cup r \cup q)$. However, under normal circumstances the fact that the plane is on the ground and has just completed a descent does not imply a Go Around situation. It simply does not make it impossible.

- The fact that the pilots are attempting to engage the brakes suggests that the pilots do not wish to take off/go around. $C \rightarrow p$.
- The fact that the configuration of the flaps is such as might be used for landing implies that the plane is landing – $D \rightarrow p$
- If the main wheels are engaged with the ground, the plane is currently on the ground ($A \longleftrightarrow F$).
- The intention to take off requires that the configuration of the flaps are changed and that the pilots are not attempting to engage the brakes — $p \rightarrow (\overline{C} \cap \overline{D})$.

Where certain things are defined as 'certain', logically speaking, it is possible to define statements as 'true' or 'untrue' within the framework. For example, were we to suggest that the plane was landing (in terms of intention) but that the current reality is that the brakes are not being used and the flaps are not correctly configured, then either the analysis of the intention was wrong, or the current reality of the situation is non-optimal and should be altered if possible. If many of the facts suggest one intention, as in this case study, then perhaps the concept of 'fuzzy logic' comes into play to some extent —most of the facts suggest one thing, so the most likely solution is that solution.

Fuzzy logic is a superset of conventional (Boolean) logic designed by Dr Lotfi Zadeh[75], that has been extended to handle the concept of partial truth – truth values between "completely true" and "completely false". Zadeh suggests that 'fuzzification' as a methodology could be applied to generalise any theory from discrete values to a constant spectrum.

12 Language and interaction

If the first layer of abstraction between the environment (the plane and those external variables that have relevance in this situation) and the end users is telemetry, the eventual question of interface remains.

It is already clear that whatever method of interaction chosen, it must make use of a 'language' sufficiently rich to allow concepts such as the distinction between 'factual' and 'facts that result from analysis' to be clearly represented. This, of course, must be to the satisfaction of both human and computer. Each form of information has its place, and both user and software should share the same views on its status. If this is not so, one risks misinterpretation of, for example, a prediction as a certainty, which is potentially dangerous.

Both the human-computer interface and the agent-agent interface need to be considered. Although each element of the agent system is likely to have, to some extent, its own specialisation and therefore boundaries —which is to say, each agent is required to understand only a subset of the overall language — this is not true for the former interface. Each interface requires, for definition, that the objects and their relations to each other are designed.

This problem is one of the special requirements for knowledge-based systems[34]; since these are designed to operate on and make use of statements in a formal knowledge representation, and exchange knowledge, there is a real need to define a standard 'language'. Several proposals have been made for standard knowledge representation formats and agent communication languages. Gruber suggests that as agreements must be established about language (eg, shared assumptions and models), ontologies[4] may play a software specification role.

Gruber's design criteria for ontologies are reproduced below, since they are relevant to the projected design;

[4] An ontology is 'an explicit specification of a conceptualization'

- Clarity - definitions should be objective, defined as logical axioms wherever possible. Formalism is a means to this end.
- Coherent - consistent with the definitions
- Extendibility - One should be able to define additional terms for special uses, that do not require the redefinition of the existing terms
- Minimal encoding bias - Representation choices should not be made purely on the grounds of convenience within a given system
- Minimal ontological commitment - It should make as few claims as possible about the world being modelled

However, the above has not yet been applied to this problem.

The definition of an interface language implies that the relationship between the computer and human pilots be described. As previously noted, the designers of previous human-computer interfaces for similar applications have often avoided this to the fullest extent possible, preferring to simplify the interface to the point that the idea of "Computer as reasoning tool capable of independent action" is quite lost in favour of the description of the computer in terms of resultant phenomena (eg. a given button activates a given system, but it is little noted that the system in question is acting as part of a larger superstructure of software —it is a 'function' to be switched on or off, rather than a small part of a larger, more complex system). Jacob [39] mentions this as a difference between real human communication and human-computer communication, noting that "In a direct manipulation or graphical interface, each command or brief transition exists as a nearly independent utterance". It would be unrealistic to expect pilots to understand the details of the system, as it is currently implemented, as it has been engineered precisely to avoid the idea of any underlying complexity. Altering the interface to the system has a deeper effect than altering the way it looks, also acting to change the users' fundamental perception of a system (Ledgard [42] 1980).

The author contends that by effectively hiding some of the rules that govern the system as well as part of its nature, the system may sometimes react in inexplicable ways. The 'design metaphors', the concepts with which the pilots are trained to understand the system, may break down —they may be inadequate to explain all the phenomena observed.

There are two apparent solutions to this :

- Design a better simplified interface, or using the given interface, extend the system itself so that it more exactly agrees with the pilot's current ideas of what the system does (the Do What I Mean direction)
- Look beyond the idea of representing a computer as a set of readily available (apparently disconnected, that is, internally separate) functions

Yet a system with more interrelated parts is, conceptually, more complex. It is therefore suggested that both approaches are used, for separate purposes.

Both of the systems described below are of course different interfaces to the same underlying software, intended for separate purposes. At the present time the line between them is somewhat arbitrarily drawn, due to a lack of sufficiently detailed data at the present time on standard operating procedure/failure modes.

Firstly, there may be a use for a system that provides information on demand, which is to say a passive system. This would be useful for pilots who wish to make use of standard functions, check information manually, review some facts or the status of the plane, and so on. Such a system is not far from the traditional point-and-click interface, although simply implementing a traditional GUI might be somewhat shortsighted, since once again the question of mutually shared appreciation of context is vital. The computer is capable of providing factual information, analysis, predictions and volunteered information, but the distinction of a given piece of data into factual, analysed, or predicted information is a relatively subtle one. A passive system is, per se, incapable of volunteering information effectively in a situation where time constraints apply.

This system alone therefore contains a number of interesting design issues :

- The 'language' it uses must be sufficiently rich to communicate as above
- The agent system is in possession of a bewilderingly large quantity of data, much of which has no obvious meaning to the human without analysis.
- Other traditional interfaces already exist that indicate much of this data in some way. Can they coexist?
- As an interface to an agent-based system, it has the same problem as any such; agents communicate in between each other, typically, using a language like XML/RDF over TCP/IP[27]. The human interacting with that system does not. Though the internally used language may be rich enough to permit full communication between the agents, it is unlikely to be suitable for the human-computer link. So a simplification is made; the various concepts and actions embedded in the language are extracted and placed on a simpler interface for the user to manipulate at his/her leisure. The effect of this, however, is very probably a decontextualisation of that information. Although it is seldom considered as such, information output from the computer may be as out-of-context as isolated input.

Secondly, there is a use for a system that volunteers information. This is another issue, and quite different (although linked) to the first. Such a system must be able to give information in a modality sufficiently well-supported that the pilot(s) immediately take notice of it. Speech output, for example, might be effective. Visual output is comparatively passive, since it attends one's attention. This requirement is at the heart of the paper. There is little benefit to be gained from, essentially, adding an additional signal lamp to the cockpit that, if read, would indicate that another signal lamp has been ignored; if this intelligence is not communicated to the pilots at the earliest time possible then it is simply a waste of computing time.

However, the best way of communicating important information is far from obvious. An interesting thought-experiment is to imagine that you, a human being not recognised by

the flight crew as part of their team, had just noticed a problem with the plane that you felt should be communicated to them. How would this best be approached?

If you, a passenger, approached the copilot directly and attempted to persuade them of your information, particularly assuming that a crisis/time-dependant situation is involved, the situation would be awkward at best. Perhaps a better strategy would be to ask a steward or stewardess to carry the message to the flight crew, and depending upon your persuasive abilities the information would eventually be transmitted. Compare this to the situation in which the copilot noticed the problem. There is only one stage of communication involved, and the information comes directly from a trusted source. From this, it might reasonably be concluded that information alone is not sufficient. Equally important is the perceived source of the information and the trust that is put into that source.

An effective way of approaching this design, therefore, is to consider the team structure of the flight crew and the ways in which they interact/share data amongst themselves. The procedures between members of the flight crew and their own conceptualisation of the relationships between themselves, the roles each crew member should play, may allow a suitable place for an additional (artificial) member of the team to be added. This, therefore, becomes the study of human-computer *collaboration*.

Human-computer collaboration has previously been focused on by a number of researchers, including the MERL team[48], who focused on plan recognition, the process of inferring intentions from actions. They note that plan recognition has proven difficult to design into practical systems due to its inherent intractibility in the general case. Fortunately, the specialised and restricted nature of the tasks with which a cockpit system is involved may alleviate this problem, since it has the effect of severely constraining the span of 'sane' plans. Since each interface is provided with a 'role specification', any plans that are outside the normal functioning of the system are those with which its involvement is unlikely to be desirable. There are only three modes involved, flight, takeoff and landing, of which the second and third are most volatile.

It is also worth noting that communication between members of the flight crew is itself largely designed around the development of mutual understanding. Requests for feedback are common ("You happy?", or "Got it?"). In the case study at which this report began, much of the confusion of the incident arose from the captain's lack of feedback to the others — he neglected to specifically state his intention to abort the takeoff/go around, as a result of which a period of time occurred in which the crew were essentially attempting to achieve opposite results.

An additional piece of information that influences the design of the system involves the current roles of pilot, copilot and second copilot. Quoting from the Pilot and Flight Engineer Career Overview at avjobs.com[5], "The position of flight engineer, or second officer, exists only on some of the large jet planes. Smaller airliners-as well as the newest large aircraft-have only a two-person flight crew, consisting of the first officer and captain. Functions of the flight engineer include inspecting the aircraft and overseeing fueling operations before flight. During the flight, the flight engineer monitors the performance of the engines and cabin pressurization, air conditioning, and other systems." The second officer is in a very similar position to that which the proposed systems attempt to support (or play). In cases where a human second officer is on the flight, perhaps the passive system should be used as a support for his/her usual tasks. When there is no second officer, their role is perhaps a blueprint on which the activities of the active system could be modelled.

[5] http://www.avjobs.com/careers/pilot/Pilots_FlightEngineers.asp

13 Interface hardware/software options

There exists a wealth of research on human-human collaboration using computers as a medium, that may provide some hints as to the more helpful alternatives.

Possibilities include voice input/output, traditional graphical output, standard input devices such as keyboards, mice, and touch-screens, position/motion of limbs and full-fledged virtual reality. Each of these have relative strengths and weaknesses, depending upon the application. Since the intention is to 'increase the bandwidth' of communication, it is tempting to assume that the more modes available, the better. However, this again is likely to depend on application; for a passive system, is a multimedia interface necessary or desirable? For an active system, however, one designed to take an active role as part of collaborative activity, the intuitive conclusion is that a multimedia interface is potentially helpful.

Certain among these possibilities may be discounted, particularly virtual reality. The ideal use of virtual reality is as a method to provide immersive access to a virtual environment, frequently in order to support actions that cannot easily be completed in the real world (such as interacting with models of as-yet-unbuilt cars [62] or models of molecules[67]).

Disregarding for a moment the relative success of these input/output systems, it must be noted that the conditions in an aircraft cockpit are not optimal for all interfaces that may work in other situations. In situations where a 'watchdog' program is likely to volunteer information, for example, time may well be short, and therefore the quickest possible way of transmitting information is preferable. However, stresses are also high in such situations, suggesting that perhaps causing abrupt and 'unnatural' interruptions is an unacceptable strategy.

14 Approaching Implementation

A number of areas have been noted as interesting targets of further research. In the cadre of this particular problem, the next steps are :

- Definition of a suitable agent/distributed architecture in detail
- Definition of an intra-agent language suitable for this application
- Description of a logic by which a given system may be parametrised

There are also the further issues, such as the nature of teamwork itself, and the most advantageous way of approaching the interface design issue that a truly autonomous agent presents - where an autonomous agent is one that is self-directed, dealing with incomplete and imperfect information in order to make decisions and put those decisions into action. Teamwork itself raises the social issues, such as personality compatibility, implied competence, and trust.

Further research into appropriate interfaces is particularly of interest - particularly the question of supporting human-computer collaboration. This appears to have previously been discussed for text-mode interfaces, for example, Lesh, Rich and Sidner's work on using plan recognition focussed on application of their plan recognition algorithm to an e-mail program. The integration of such a system into a traditional graphical interface is an interesting experiment, not least because it changes the nature of the communication between human and computer, replacing a previously required statement of each subgoal in detail with a more 'natural' dialogue.

However, there appears to be comparatively little research available on the subject of human-computer collaboration using less traditional interfaces. There is also the question of the distinction between 'passive' and 'active' interfaces that has been somewhat arbitrarily made in this paper. Referring once again to Svanæs' paper, the discussion of the phenomenology of Merleau-Ponty, one notes Merleau-Ponty's examples of the body's ability to adapt and extend itself through external devices. Firstly the example of the

blind man's perception of the world using touch — a stick — where the tool becomes in a sense an extension of themselves, part of their body.

The second example, an organ player, is intended to illustrate how external devices may be internalized through learning. It is this example that is the more interesting from the perspective of this discussion, the level, and nature, of the involvement that a computer user has with the equipment that is used.

Quoting Svanæs once more, "Through skill acquisition and tool use, we thus change our bodily space, and consequently our way of being in the world". At this stage, although this is largely a speculation, it appears that computers have typically taken on one of a number of roles; firstly, the 'invisible', such as the microchip in the refrigerator, and secondly, a role in which the computer's functions are represented to the user as if it were a simple mechanical system. Evaluation systems such as GOMS are the zenith of such a mentality in HCI. Neither of these designs are 'wrong' in all applications, indeed, for a simple system that requires very time-constrained usage, the latter role is quite appropriate. A third role is that held by many modern GUI systems, semi-passive systems that attend user input but sometimes provide some direction to users (in a similar way to that described by Lesh et al.).

The comparison between these systems and other, newer systems (for the above list is unlikely to be in any way complete), is interesting in itself. It brings to mind the controversial topics, dating from the dawn of the desktop system or before, of graphical systems versus the command line interface. Perhaps the classification of a 'passive' system is something like that of the musical instrument in the example of the organ player, a device that provides a relatively intuitive way in which to access a number of functions, on the request of the user. Such an instrument is predictable, unlikely to produce unlooked-for effects once the operating rules are learnt, although this may not be a trivial process. An 'active' system, on the other hand, is designed in order to point out the unforeseen. The actions of such a system are not those of visible cause-and-effect — information volunteered as

output from the system is as a result of certain inputs, but the user's lack of awareness of these inputs means that they are unlikely to be able to see the correlation. The same is true in current systems, of course, but this is most frequently as a result of a breakdown.

The problematic issue with a phenomenological viewpoint is that it is a measure of effect; it does not permit a more complex cause-effect link. Returning to the first days of HCI's involvement in aeroplane visual displays[57], the purpose of machine layers between humans and the world is to 'permit the machine to do jobs that our senses cannot undertake; - whetherbecause we cannot perceive the information, sense it accurately, or because the sources of information are so diverse that it is impossible to undertake the task without bringing the information together in some convenient central location'. This was many years before it became possible to trust almost all functions of an aeroplane to a computer — in recent years it has become unusual to find pilots controlling modern aircraft directly for any period of time. Equally, the perception of a machine with any great complexity and capacity for planning, the 'weak' intentionality previously discussed, was as yet far away.

Had this article been written at a later date, or perhaps revised, the author might have added that another task of the machine is to present its understanding, clarify, and justify its decisions. An interface which presents phenomena without explanation is useful, for those situations where the machine's purpose is simple enough to require no explanation or justification, or for a system that is 'perfect'. However, an aeroplane's avionics system is imperfect by definition — it operates within and depending on an environment that is inherently rather unstable, not entirely predictable, and by manipulation of a large number of variables of some complexity.

15 Review of safety-critical system design and requirements

It is evident that any system such as aircraft avionics is by no means harmless; a malfunctioning system - or a system that failed to work effectively in the way that the crew of the aircraft expected - could cause a serious, even fatal accident. Quality-control is obviously extremely important, to ensure safety and correct functionality of the system. This means, of course, that an avionic system is one of a class of *safety-critical* systems.

Safety-critical systems, often called *safety-related* systems, are systems whose operation has a direct influence on the safety of their users and on the public, or more rigorously, given the definition of safety shown here:

> Safety is a property of a system that it will not endanger human life or the environment.

safety-related systems may be defined as follows:

> A safety-related system is one by which the safety of equipment or plant is assured.

(nb. Much of this is taken from Storey, 1996 [65]) Before looking closely at system design, it is important to examine the 'special requirements' that safety-critical system design imposes on the designer. Later on, it is worth examining these requirements in the light of criticism aimed at the techniques suggested; however, let us first introduce them as they are presented to the designer of such systems.

15.1 Introduction

System design, in the sense of safety-critical systems, involves not only the software but the hardware on which it runs - and it is normally considered all the better to avoid complex computer systems entirely in favour of electronics of a more dependable nature. Computer-based systems are fast, low in power requirements and extremely small. Whilst modern digital devices themselves are reliable and flexible - and therefore cheap to revise

when necessarily - they remain complex. There is a great deal of necessary components required for a computer system to operate, and many more for a software system to operate. Better, when possible, to make use of the simplest system available, one with few possible modes of behaviour, and composed of simple and reliable components rather than the several thousand composants of even the simplest computer. However, this is frequently impossible or undesirable. In either case, however, a rigorous system has been documented by designers working in related industries which aims to assure the quality of such applications.

By *quality*, many issues are represented. The primary safety of the system, in terms of the electronics of which it is composed and their failure modes, such as electrical fires or short-circuits; the functional safety of the system, equipment controlled by the computer and the potential errors that may be caused, and thirdly the indirect safety of the system - the potential results of computer failure or production of incorrect information.

Autopilot systems, being directly in control of the equipment controlled, are naturally referred to as control systems; systems that oversee their correct functionality are *protection systems*, or *watchdogs* in the vernacular.

All computer-based problems should begin with an investigation of the safety implications (MoD, 1991). Once the safety implications are enumerated, it is possible to allocate to a project an integrity level that reflects its level of criticality. This determines the methods used for the design, construction, and testing. Thus, each project begins with a hazard and risk analysis.

A typical lifecycle development model is shown in fig.1.

This lifecycle model is only an example; many models exist that work in more or less similar ways. Each has a dissimilar emphasis, however — some concentrate on exact planning of development timescales, others on cost.

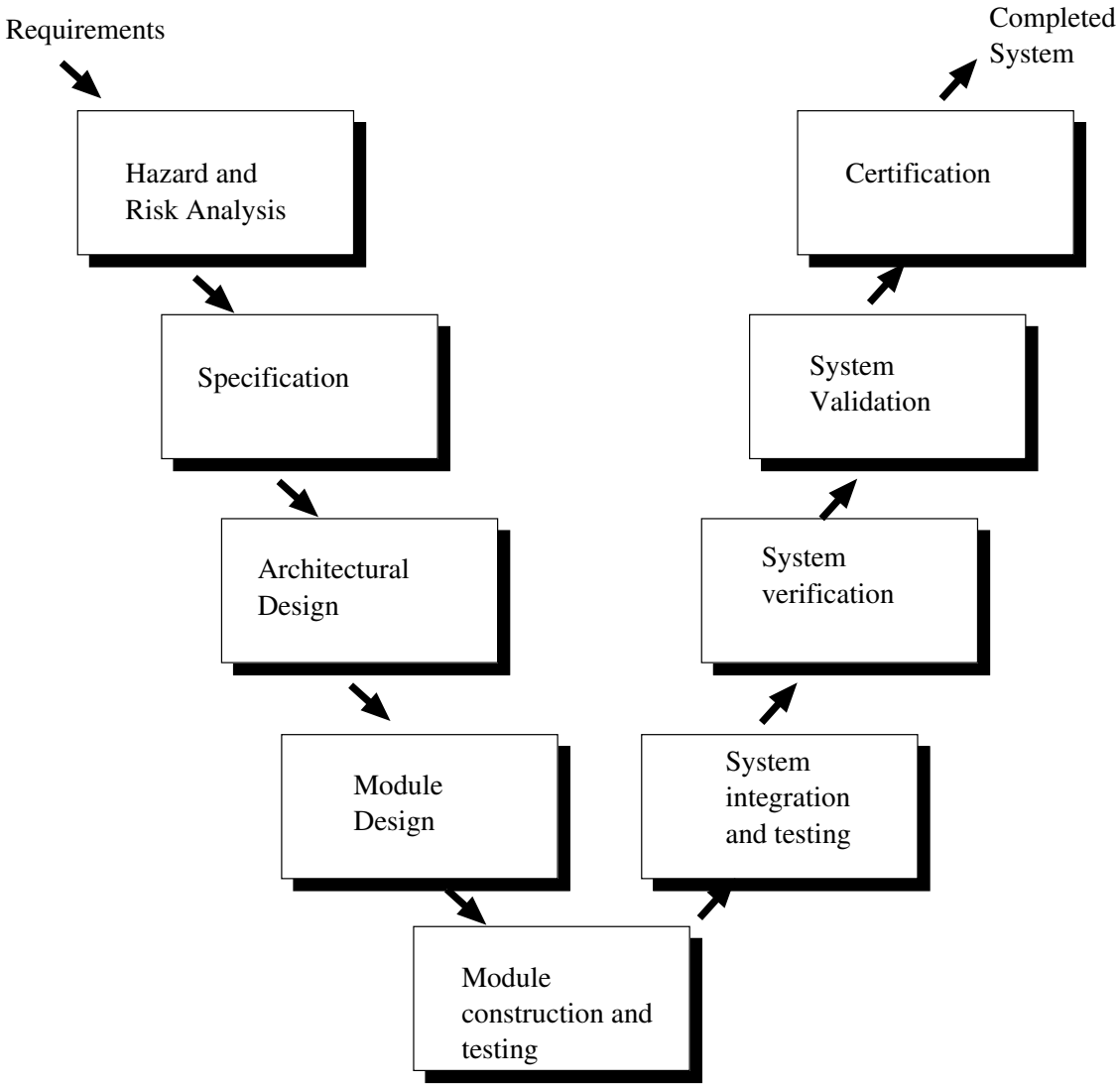


Figure 1: Typical lifecycle development model

Note that one begins at the system requirements, that represent an almost abstract definition of what the system should do. These are then formalised into a functional requirements document. This, a definition of the system's intended behaviour, may not be complete or accurate. Therefore, they are constructed with a great deal of attention to detail.

Hazard and risk analyses are then performed to identify potential dangers in the system and allocate an overall level of integrity. A statement of the safety requirements, noted

above, are then stated - the parameters of the system's behaviour, if it is to function safely. These then become, effectively, part of the system's functional requirements.

Following the development of the specification, the system's top-level architecture is designed. The system is designed to make maximal use of the advantages of both hardware and software. The software is split into modules, each one of which is specified separately such that they may simply be verified. This is followed by system integration, the assembling and testing of modules into the overall system.

This step is followed by the process of verification and validation;

> *Verification* is the process of determining that a system, or module, meets its specification
>
> *Validation* is the process of determining that a system is appropriate for its purpose

Verification is done at the module level - the checking of the behaviour of a module by comparison to its specification. Validation takes place following the integration of the system, to ensure that the system meets the requirements of the user. Therefore, the process of validation is not a matter of checking the behaviour of the system against the specification - it is a matter of checking the correctness of the system and the specification by comparison to the actual purpose of the system.

Finally, certification is completed; standards and guidelines exist for this purpose.

It is generally suggested that the lifecycle shown here is usable for any system.

15.2 Preferred characteristics of safety-critical systems

Generally, there are a number of preferred characteristics for safety-critical systems :

Reliability - the probability of a component, or system, functioning correctly over a given period of time under a given set of operating conditions.

Availability - the probability that the system will function correctly at any given time.

This is often referenced as unavailability; (1−i availability).

The *integrity* of a system is its ability to detect faults in its own operation and to inform a human operator.

This is often used as a synonym for dependability. However, dependability has a distinct meaning that should not be neglected.

Dependability is a property of a system that justifes placing one's reliance on it

Data integrity is the ability of a system to prevent damage to its own database and to detect, and possibly correct, errors that do occur.

System recovery is also important, so that systems can be restarted in case of failure for any reason, quickly and effectively.

The duration of the fault situations are important. Maintenance is the action taken to retain a system, or return a system to, its designed operating condition. Maintainability is the ability of a system to be maintained. It may quantatively be treated as the mean time to repair (MTTR).

15.3 Faults exclusive to software

Typically, software programs are almost certain to contain bugs, or programming errors. As the old joke goes :

> Every software program has at least one excess line of code and at least one bug. Therefore, by a process of induction, every program can be reduced to a single line of code that doesn't work.

These faults may for example be :

- specification errors
- code errors, misspellings
- uninitialized variables
- threading faults
- insufficient memory
- logical errors
- stack/buffer over/underflows

Software faults may be removed as they are noticed. However, practically speaking, even the simplest software is dependent on many thousand, or even hundred thousand, lines of underlying code. Since such complexity is involved, it is extremely difficult to remove every bug, and almost certainly impractical on any production timescale. Therefore, software faults are typically tolerated.

15.4 Analytical techniques

In order to assess the safety of a system, it is necessary to make use of various techniques - failure modes and effects analysis, formally known as FMEA, hazard and operability studies or HAZOP, and fault tree analysis (FTA). Any potentially harmful situation is called a hazard, defined formally as :

A situation in which actual or potential exists to people or the environment.

Several studies exist that focus on problems associated with computer-based systems; Neumann and Kletz[52] [45] are excellent examples. Typically, one or more of the following hazard analysis techniques are used; failure modes and effects analysis, effects and criticality analysis, hazard and operability studies, event tree analysis and fault tree analysis.

Typically, FMEA involves a consideration of the effects of failure in any component of the system under analysis. HAZOP is essentially characterised by 'thought-experiments' such as imagining the effect of parameter changes on the system. This may be done most effectively by using computer-based techniques; however, there is a heavy reliance on the tester asking the correct question.

Event tree analysis and fault tree analysis are particularly interesting for our purposes. Event tree analysis is essentially an event walkthrough, which works well with a small, simple system (or completed on one module at a time of a larger system). However, for a complex system it rapidly begins to produce very large trees. Probabilistic analysis may be applied to the tree in order to summarise the likelihood of any given situation. Fault tree analysis, by contrast, begins at the 'scene of the crime', the fault itself, and works backwards to the cause.

The symbols used in fault tree analysis are provided in IEC 1025.

15.5 Measuring the severity of risk

Mathematically speaking, risk is defined as a combination of the frequency or probability of a specified hazardous event, and its consequence. That is to say,

$$Risk = severity * frequency \tag{1}$$

Severity, in this case, is typically defined as the number of people killed. Frequency is the number of times that such a situation occurs. Both units are typically given per year (eg;

a situation that could kill n people occurs m times per year. Therefore, the risk is equal to $n * m$).

15.6 Allowable risks

Standards exist that provide a specification of allowable hazard probability class for each of the hazard severity categories (FAR, 1993, JAR, 1994). Essentially, a maximum probability is specified for each class of failure effects. Civil aircraft equipment manufacturers, currently, must achieve the requirements set out below. It is important to note that the standards do not demand a zero failure probability; to do otherwise would be unrealistic. The relationship between effect severity and allowable probability for civilian aircraft is shown below :

Category	Severity of effect	Maximum probability per operating hour
Normal		10^0
		10^{-1}
Nuisance		10^{-2}
Minor	Operating limitation; emergency procedures	10^{-3}
		10^{-4}
Major	Significant reduction in safety margins; difficult for crew to cope with adverse conditions; passenger injuries	10^{-5}
		10^{-6}
Hazardous	Large reductions in safety margins; crew extended because of workload or environmental conditions. Serious injury or death of a small number of occupants	10^{-7}
		10^{-9}
Catastrophic	Multiple deaths, usually with loss of aircraft	10^{-9}

15.7 IEC 1508

The draft standard IEC 1508, "Functional safety - Safety related systems", provides integrity levels in failures per year. Remarkably similar to the accident risk classes for military systems, it is intended to 'cover those aspects that need to be addressed when electrical/electronic/programmable electronic systems are used to carry out safety functions [...particularly...] those of any software forming part of a safety-related system or used to develop a safety-related system'. It defines four risk classes, I to IV, where IV is the least serious :

Frequency	Consequences			
	Catastrophic	Critical	Marginal	Negligible
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

These risk classes must be interpreted as follows :

Risk class	Interpretation
I	Intolerable risk
II	Undesirable risk, and tolerable only if risk reduction is impracticable or if the costs are grossly disproportionate to the improvement gained
III	Tolerable risk if the cost of risk reduction would exceed the improvement gained
IV	Negligible risk

Target failure rates are also provided within this specification :

Safety integrity level	Continuous mode of operation (probability of a dangerous failure per year)	Demand mode of operation (probability of failure to perform its designed function on demand)
4	$\geq 10^{-}5$ to $< 10^{-}4$	$\geq 10^{-}5$ to $< 10^{-}4$
3	$\geq 10^{-}4$ to $< 10^{-}3$	$\geq 10^{-}4$ to $< 10^{-}3$
2	$\geq 10^{-}3$ to $< 10^{-}2$	$\geq 10^{-}3$ to $< 10^{-}2$
1	$\geq 10^{-}2$ to $< 10^{-}1$	$\geq 10^{-}2$ to $< 10^{-}1$

In effect, however, the IEC 1508 includes integrity levels that cannot be dependably demonstrated. Thus, it is considered inappropriate to specify systems requiring an integrity above this level (that is, safety integrity level 4).

However, engineers of certain systems have claimed rates many times better than that represented in safety level 4.

15.8 Defining and characterising system faults

System faults are any failure that occurs within a system; for example, the failure of a hardware component or the failure of a software component, whatever the reason; communication faults, mistransmitted information, or a broken switch are all examples of system faults.

The aim, in handling such faults, is to ensure that the fault is not sufficient to cause the system as a whole to fail — that the fault does not cause an error, or, worse, system failure. System failure is the deviation of the system as a whole from its required operation.

Faults are generally divided into random and systematic faults; random faults are simply events with no particular reason, whilst systematic faults are events that occur due to a

reason. For example, GPS's low coverage in the North may give rise to systematic faults in hardware that makes use of GPS input, since the orbits of the GPS satellites themselves determine the availability of the system, together with their height above the horizon.

Faults cannot be eliminated entirely. However, the system may be made sufficiently robust that faults do not affect the running of the system, by detecting faults, cross-checking information, for example; by avoiding the introduction of faults, particularly systematic faults; by removing the faults, by a process of verification and validation, application of formal methods such as PROVER, and reverse-engineering against the specification in order to check the extent to which the system implements that specification[55].

15.9 Combatting system failure

According to the above, system failure is inevitable — the work of the engineer involves reducing the frequency of failure incidents to an acceptably low standard. As safety-critical systems have become more popular — and more necessary — engineering methods have been proposed to do just that. Various methods have been suggested, including 'safer' software development methods that require rigorous design, code generation, verification and validation systems at each step of the way. Such design methodologies go some way to explain the interface issues found on planes, however, since the recently published safety-critical design documents mention both interface design and user-centered design/feedback in a cursory manner. Ergonomics is more frequently considered as part of the engineer's design problem, although more interest historically has gone into functional than conceptual issues.

15.10 Preferred software development methods for safety-critical design

A great deal of research has been devoted to the choice of programming languages considered to be acceptable for such systems - for example, Carré 1990 and Cullyer 1991. Particularly important are the functional characteristics of the language, the support tools

available, and the expertise available within the devleopment team. Carré suggests that the six factors that are most relevant to the choice of programming languages are :

- *Logical Soundness:* the existence of an unambiguous definition of the language
- *Complexity of definition:* Do there exist simple, formal definitions of the language features?
- *Expressive power:* can program features be expressed easily and efficiently?
- *Security:* can violations of the language definitions be detected before execution?
- *Verifiability:* Does the language support verification (code consistency with specification)?
- *Bounded space and time requirements:* is it verifiable that time and memory constraints cannot be exceeded?

These are non-trivial requirements; most programming languages are typically designed towards other, commercial, requirements, such as ease of use or short development time. In effect, a comparison by Cullyer (1991) of computer programming languages shows that the more popular programming language, C, is particularly inappropriate for safety purposes.

	Structured Assembler	C	Coral 66	ISO Pascal	Modula-2	ADA
Wild jumps	*	?	?	?	?	*
Overwrites	?	X	X	?	?	?
Semantics	?	X	?	?	*	?
Model of maths	?	X	?	*	*	?
Operational arithmetic	?	X	X	?	?	?
Data typing	?	X	?	?	?	*
Exception handling	X	?	X	X	?	*
Safe subsets	?	X	X	X	?	X
Exhaustion of memory	*	?	?	?	?	X
Separate compilation	X	X	?	?	*	*
Well understood	*	?	?	*	*	?

The more popular programming languages for safety-critical applications in general are ADA, modified to produce a safe subset, or a similarly modified Modula-2. Certain avionics systems are written in C++. However, the majority of these are non-critical systems whose failure produces no more than passenger dissatisfaction. Structured assembly language is also relatively popular. Rowe (1994) showed in an informal survey that the development of the Boeing 757-767 included the use of a total number of languages ranging around the 140 mark.

One note of some relevence later in the paper; Boeing make use of the language Prolog to build their own custom rule languages, as well as to advise their engineers, manufacturing and field personnel as to the best manner to assemble specialised electric connectors.

The Boeing 777 aircraft may use the Alsys CSMART verified Ada kernel; however, even this is designed specifically for systems of relatively low criticality.

15.11 Overcoming the troubles of software; fault-tolerance

The reliability of a system may be boosted either by the use of pure software methods, or by fault-tolerant architecture. Chen and Avizienis, 1978, suggest making use of a technique known as N-version programming - making several different implementations of the same specification. These programs, ideally, should all produce the same results. The different versions are run for each situation, either sequentially on the same processor or in parallel on a number of different processors. Where N=2, a self-checking pair is effectively created; the outputs go to a comparator, and in the case of a mismatch, the system signals failure to the output. This can be extended to any number of modules and effective complexity. However, a system with an odd number of modules is desirable since the system may be able to implement a form of voting to work out the results. In practice, cost makes this impractical.

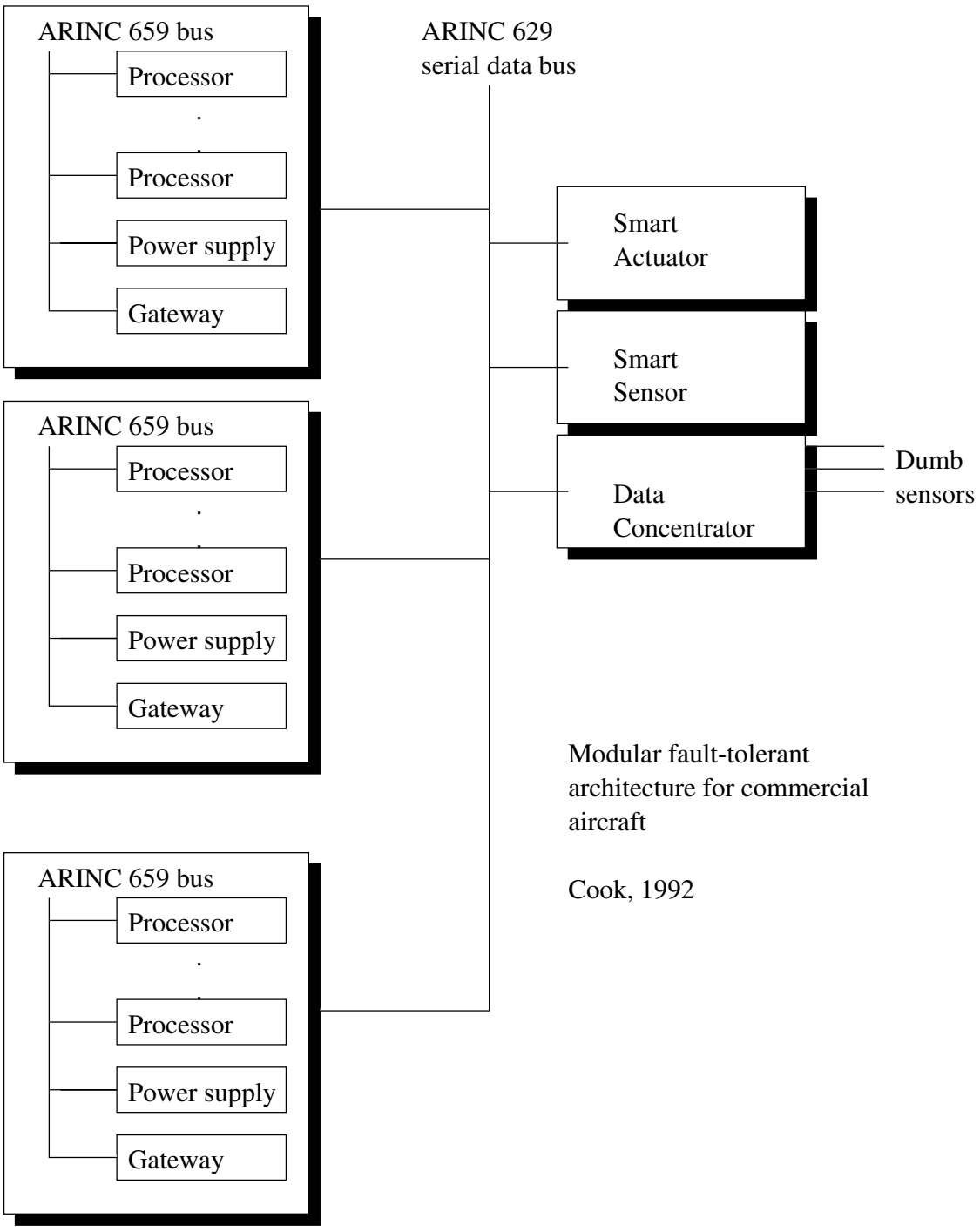
In the case of larger, voting systems, N-version programming is effective not only to make visible transient hardware faults but also to safeguard against systematic faults associated with any given implementation. However, should a fault exist with the specification or elsewhere in the shared program design, it does not guarantee that this fault will be recognised, much less solved. Airbus A330 flight control systems are considered critical enough to make use of such a technique.
The 'recovery block' system is also used. This is a method by which each module may be made self-testing, using acceptance tests that include checks for runtime errors, reasonability, excessively long execution time and mathematical errors. This might be characterised as below, in the following pseudo-code :

```
Ensure     Acceptance_Test
By         Primary_Module
Else   by  Alternative_Module_1
Else   by  Alternative_Module_2
  ⋮
Else   by  Alternative_Module_n
```

There remains the possibility of hardware faults, however, in such systems. In effect, N-version systems (or systems of N identical modules) are usable for the majority of situations.

15.12 A real-world example of fault-tolerant avionics systems

Cook (1992) demonstrated a prototype standardized, modular computer architecture for use in commercial aircraft. The system includes a great deal of redundancy.



Modular fault-tolerant architecture for commercial aircraft

Cook, 1992

16 Engineering of artificial intelligence software in general

Throughout this report, the autopilot system has been designated as a member of the notorious class of software typically described as 'A.I.'. Without introducing the philosophical question of whether an AI system may be described as 'intelligent' in any meaningful sense - at this stage - it is worth looking at what the designation means in real terms.

A reasonable definition for *artificial intelligence* comes from Partridge (1992).

> *Artificial Intelligence:* is that field of computer usage which attempts to construct computational mechanisms for activities that are considered to require intelligence when performed by humans.

A more complete definition (*ibid.*) is also provided.

AI Problems	Conventional software problems
1. Incomplete, performance-mode definitions	Complete, abstract definitions
2. Solutions adequate/inadequate	Solutions testably correct/incorrect
3. Poor static approximations	quite good static approximation
3. Resistant to modular approximation	Quite good modular approximations
5. Poorly circumscribable	Accurately circumscribable

There are several points brought up in this table - AI problems are ill-defined. An AI system is defined in the first quote as one which attempts to construct computational mechanisms for activities that are considered to require intelligence when performed by humans; it is therefore a system which is described by looking at human performance when faced with a given problem. In this sense, it is difficult to compose a rigorous specification for an AI program. Although the desired output is, typically, quite well defined, the mechanism that an AI system is intended to implement is only designed in terms of a given input and output.

Since the AI system was described in terms of the desired output, it is also remarkably difficult to test. It is possible to give certain inputs to a system and see whether the output appears accurate - but often, since the problem is ill defined, the solution may not be entirely clear either. Many AI problems may give rise to several plausible solutions for a given input - one cannot easily test a given solution's correctness in any rigoroous sense.

Many AI systems also provide poor static approximations - their solutions may change dynamically. Most places in which computer software are seen are those in which static approximations are available.

Modularisation, frequently used as a useful tool for rationalising otherwise complex problems, has a tendancy to fail in the case of AI problems. This is typically because they are deeply embedded in context, the stripping away of an element therefore producing a situation in which context is removed and the information under examination therefore becomes relatively meaningless, or possibly misleading. Perfect examples for this can be seen in natural language processing (NLP) research, where tonality, phonetics, syntactics and semantics are so tightly coupled together that each may effect the perceived meaning of a phrase.

This is not to say that modularisation in artificial intelligence is impossible. One successful manner of modularising such problems is the 'blackboard architecture', which is relatively effective. Alternatively, modularization may still be successfully applied by making use of various strategies, such as a preference for object-orientation over structured methology.

Finally, AI tends to be massively coupled with context. Unlike classical computing problems, which essentially require nothing but a mathematical system in which to function, AI tends to be remarkably dependent on the user's mood, intention, bias and prejudices.

Given the above, it is easy to see that the use of AI does not easily mix with safety-critical design processes. Is it, nonetheless, worth examining?

16.1 Design of AI systems

Dijkstra and others share a dismissive view of AI in general, considering that "the only problems that we can really solve in a satisfactory manner" are "problems that admit nicely factored solutions". This is certainly a compelling point of view, given that the very nature of the situations in which AI is likely to be implemented are, in themselves, hostile to a 'clean' solution. However, it has been suggested by Partridge (1992) that a software system development methodology tailored to the nature of AI problems may provide an alternative solution.

Partridge suggests that an iterative, evolutionary approach to engineering AI software is more appropriate than the traditional methods of system design, and argues for this in great length in his work, Engineering Artificial Intelligence Software. He suggests a new discipline; exploratory system design.

16.2 AI System Architectures

There exist a wealth of proposed architectures for AI systems. Certain are designed originally as a response to a single problem, others were designed in the hope of producing a more general solution. Each one is designed with different priorities in mind. Issues of importance include :

- Scalability
- Robustness
- Reusability
- Speed
- Efficiency

Several of these issues have proven to be relatively difficult to resolve - however, many of the issues have been tackled with reference to one central model, the blackboard model (*c.f.* Fig. 2.

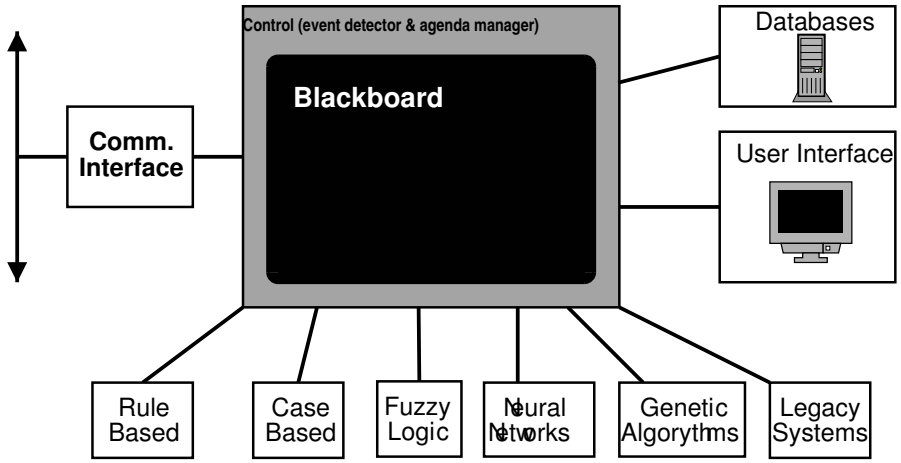


Figure 2: The Blackboard Model

This model is designed in such a way that a central repository, the 'blackboard' referred to in the title, exists, and stores all of the knowledge available - for example, facts, conclusions, assumptions and deductions. The systems that make use of the blackboard are seen as, conceptually, experts, gathered around a blackboard, each solving problems and placing the results back on the blackboard. Data also travels from and to the blackboard system via other systems, such as sensors, databases, user interfaces, and so on. In this model, the control of the system is taken by a central agency controlling the blackboard itself. This system controls the agents' access to the blackboard, managing scheduling, choosing the most promising solutions, and generally auditing the contents of the blackboard.

It is this facilitator which is the most complex aspect of the system, in a sense; each expert is in control of only its own domain. The idea of a central agency able to review the contributions of the programs and choose those worthy of inclusion confers a lot of power - and a lot of responsibility - to it. Therefore, one alteration that is proposed is to 'keep the working' - if a solution is numerically/factually correct according to the experts' best

efforts, its removal from the shared memory should be committed because that solution is:

- excessively old
- based on suspect information

Rather than working towards innate efficiency (eg. working to reduce the amount of used memory), this system relies on retaining all data that could be in any way useful, together (and linked) with the facts that the data comes from and the conclusions built on that data. This data retention increases the cost of the system as well as the number of potential faults in the hardware, but permits increasingly sophisticated agents to be built for the purposes of data mining during the flight eg. looking for multiple occurrences of similar events, marking potential trends for safety analysis purposes, and so on. An interesting introduction to the subject of data mining as such has been published as a white paper by CAASD[25].

As a design preference, it seems equally reasonable to have a separate system capable of, so to speak, wiping these obselete facts from the blackboard. However, this is only one of many modifications that might be made in order to successfully use a blackboard system in a safety-critical system, particularly one requiring near real-time performance.

The advantages of this model include a useful method for integration of disparate knowledge sources, modularity of experts and knowledge sources (since all message passing occurs via the central blackboard), flexibility, potential software reusability, and naturally, extensibility.

The blackboard model is relatively reusable, in that it has served as a basis for many different systems. However, its speed is not especially good. This is because of the fact that the blackboard model is asynchronous. Experts read information as and when it appears, sending in their 'bids' to the scheduling system. Information appears rather unpredictably, and the times of operation of the various 'experts' depends on the central scheduling sys-

tem as well as purely physical issues like the speed limitations of the computer system on which it runs.

However, research has been directed at this issue with the blackboard architecture's more traditional incarnations, such as the work of Soler et al (2002) [63]. This short workshop paper describes an architecture capable of guaranteeing adherence to a given set of temporal restraints, by making use of offline analysis, strict control over agent behaviour in the temporal domain, and the use of real time operating systems for critical elements of the design. Communication between agents is carefully bounded to fit within a single network packet, then transferred by UDP/IP as a high-speed, low band requirement solution.

Agent architectures are, eventually, very dependent on the exact problem, and for this reason certain other alterations to the model will be suggested at a later time. However, since the entire agent architecture will not be developed in practice — since an accurate simulation of the effects of a given architecture in a given situation is beyond the scope of this work, and in order for the simulation to have validity, a great deal of accurate data, hardware, and real or simulated realtime hardware would be required.

16.3 Distributed multi-agent architecture

Imagine the landing problem with respect to not one aeroplane, but several, performing a similar task in a similar location with a short time difference between each other. Each of the aeroplanes performing that task at any given time are in a good position to provide real-world measurements from that time (for example, of braking conditions, or of wheel revolutions), and are therefore in a good position to provide that information to a 'central database' (see Fig. 3. Note that the pilots themselves are typically not in a good position to provide that information, since research has shown that they seldom have a clear idea of the conditions in which they land[70].
Whilst examining this agent architecture, note that agents may cease to function, as may the data sources or transmission systems through which the data is sent. The system must

not expect the information - it is an oppportunist.

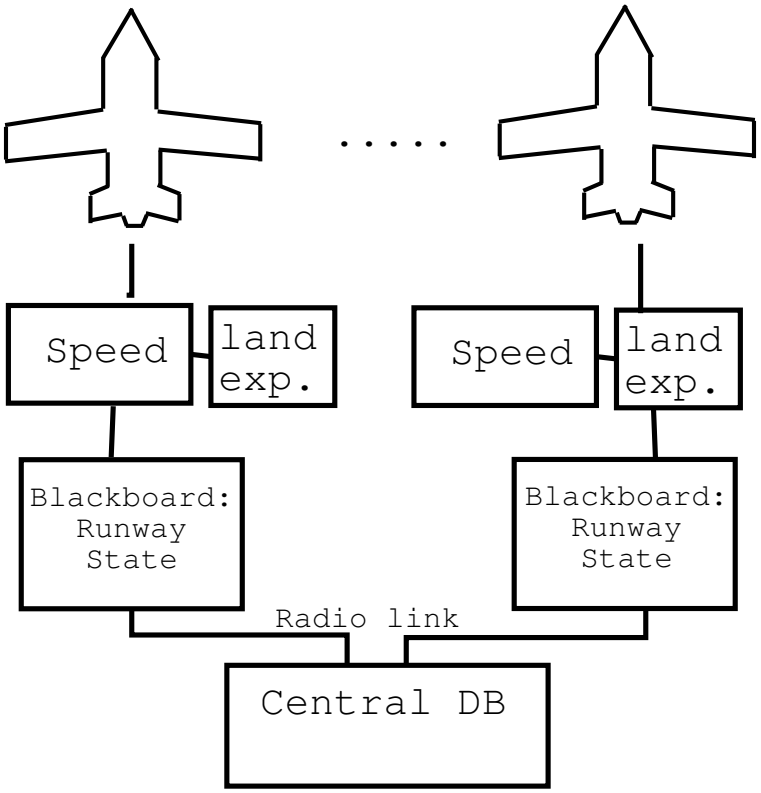


Figure 3: Multi-Agent Architecture

Thus, the plane queries the central database for information potentially relevant (for example physical context such as current weather conditions) to its current trajectory, and receives a result. Preferably, it is sure of the identity of its correspondant. The responses also come timed with a date and timestamp (if it rained last Thursday, this information must not be seen as of primary significance for current weather analysis purposes on the following Tuesday!)

17 Logic-based languages

Provided that it is possible to parametrise the system, the assertion has been made that it is possible to define the system in terms of logical assertions, predicates and relationships. It is almost certainly possible to examine such a system by using a custom-programmed system in, for example, C - using fuzzy logic, as suggested above, or heuristics. However, this is not necessarily the best way to implement the system, since the problem of logic-based reasoning has been extensively examined in the past by other researchers. Logic programming, generally, has been the object of scientific study for some time [date].

The majority of programming languages make use of two structures; data, and procedures. Rules could be considered as somewhere in between these two separate types; databases are collections of rules with no action statements and only one set of conditions, whilst a procedure is one big rule with many statements. This relationship is shown in Fig. 4.

Figure 4: Comparing data and procedure based language with Prolog

For this reason, rule-based programing languages are designed to be more appropriate for the implementation of many specifications than data/procedure based languages. This is particularly true in the case of specifications expressed as rules, which do not translate well into procedural programming.

Logic programming, as well as functional programming, represents in general a radical departure from mainstream computer languages; rather than providing a representation of the von Neumann machine model and instruction set, it is derived from an abstract model, with no direct resemblance to any particular model of machine. The concept behind logic programming involves the ideal that instructions provided to a computer should be easy for humans to provide. An incidental by- product of this, however, may be that results calculated by logic programs are made in a way — as the manipulations done by

logic programs are intended to mimic formal logic — that is relatively easy for humans to understand.

Note that the popular statement that "Programming in Prolog is equal to programming in logic" is not entirely correct. However, for all intents and purposes it is sufficiently similar.

In effect, and as it should be, given the nature of the gap that this piece of software fills in the design, Prolog shares a part of its functionality with relational database systems - a set of facts may define relations, and a set of rules may define complex relational queries in the same way as relational algebra. Logic programs may be used to define a logical database or to manipulate data structures.

In effect, logic program permits programs to be provided not in a procedural sense, but rather in terms of knowledge and assumptions solved explicitly, in the form of logical axioms. The program is therefore executed, following its initialisation by the provision of a database, in the form of a logical statement to be proven or otherwise. This statement is known as a "Goal statement".

The execution itself is an attempt, characterised by the internal workings of the specific logic programming language provided, to prove the goal statement. Typically, such a goal statement is existentially qualified - there exist some individuals with a given property.

Equally important, given the intended use of such software, Prolog demands some interesting specifics to be followed; unlike most languages, it is not possible to produce a runtime error in the traditional sense. This is because the semantics of any logic program is completely defined. However, Prolog is sensitive to its environment, in the sense that insufficient information, uninstantiated variables, for example, will provide runtime errors, as will physical problems such as lack of memory. Later versions of Prolog frequently suspend on discovery of uninstantiated variables until the values for these variables become available. This is an important consideration in view of the software architecture previ-

ously discussed, since it must include some way to date information, in order to separate new information from the old and therefore avert errors of understanding due to outdated information, et cetera.

Further issues as regards the desirability of Prolog in such systems includes the question of *predictability*. Prolog, in any of its incarnations, has not fully solved the issues that plague logic programming systems, particularly that of left side recursivity - this can produce logical loops of the type "Condition A implies condition B; condition B implies condition A", which will of course produce, effectively, an infinite loop. Although not a problem peculiar to logic programming, it is nonetheless a difficult condition to escape entirely, and it is for this reason that an implementation in Prolog should always be considered rather suspect unless tested under every possible situation. Alternatively, it is possible for the same problem to be stated in multiple ways, or indeed provided in multiple languages, such that implementation - specific issues may be removed by careful system implementation (as discussed above in 'system safety').

In a way, the suggestion is to automate tracing of the execution of a Prolog program and store the results in accessible form; a debugger with just the right verbosity, such as one that indicates the mostly-successful search tree branch. Of course, programmers are already able to use similar mechanisms by printing output, or by making use of custom debuggers — the existence of an automated way of extracting such information does, however, provide the user with a way of understanding the proceedings.

The following pages represent a demonstration of Prolog, a theorem prover dating from the early 1970s, when faced with a given query set relevent to our problem.

17.1 A sample problem

Using the Bangkok crash as a sample problem, it is possible to define a program that makes use of the Boeing databases in order to simulate the results of any given form of

landing.

Often, the databases provided are, in many respects, incomplete. They do not provide general information such as the equations that may be used to recreate them, where such exist. They are typically intended for human use rather than machine, and have as such been designed to a particular level of accuracy appropriate for that usage (of course, similar tables available that are accurate to a machine's limits almost certainly also exist, but in a way using these for this purpose would almost invalidate the point of the system — the fact that the agent is able to justify its suggestions with reference to standard reference tables). However, it is quite probable that such equations either do not exist or are of no practical use in day-to-day situations; the tables provided deal with pilots' experience with the planes and typical real-world, rather than theoretical, results. Despite this, it is possible to read them much as any engineer would, by a process of approximation and application of their own knowledge to complete the missing information — eg. by using a knowledge base to flesh out the description with a semantic interpretation.

The system, based on these basic tables of information, is naturally required to approximate a great deal. Additional refinement could, therefore, be advantageous - for example, the application of the theories that underpin aerodynamics may increase the accuracy of the system a great deal. However, aerodynamics is, as previously noted, not an exact science. It is not deterministic in the sense that absolute predictability exists - in the absence of controlled surroundings, there will always remain some uncertainty. Equally, it is useful not to set one's sights too high ; this project began with the idea that a good familiarity with Boeing's published tables might lead to a better ability to cope with unusual circumstances. Escalating the complexity of the issue, 'just because', is unlikely to help a great deal. Even if a piece of software internally calculates to an extremely high accuracy, it may be more effective to be ready to explain that conclusion in terms of a corresponding generalisation, since users do not work to the same granularity of the problem. Thinking of an expert system as a substitute for a team member familiar with a given piece of procedure and/or information, in this case, suggests that the expert system

does not have to do better than a human team member might when working to the same ends, although this would eventually be a worthy goal.

As a brief digression into the theory behind expert systems, it is perhaps misleading to refer to them as a form of artificial intelligence. More realistically, they might be seen as a method of transplanting — space- and time- shifting — the very specialised knowledge and methodology used in order to complete some expert task. That is to say, the system itself is not intelligent; it is an emulation, a reflection, of the intelligence who wrote or interpreted the tables, as well as that of the person who defines the computer interface - the provider of the knowledge being 'mined' becomes a team member, as does the designer of the interface. Computer systems twinned with expert systems do not replace people, but become conduits to others.

17.2 Proposed Architectures

The following suggestions are based entirely on adding to the existing model of aeronautic system; this is 'in line' with the aim, in that the issues identified with the original system are not of such a severe nature that the systems need to be rewritten. The autopilot systems work well as designed, to the extent of their design. Certain elements are, as previously discussed, somewhat lacking — and the addition of these elements to the existing model is described below.

17.3 A brief examination of landing

The process of simulating a landing process requires a number of formalisations and a certain amount of analysis. These are described as simply as possible, using the most basic approximations, and the process of landing described.

At the point of landing, given the speed of the plane, direction, or possibly velocity vector, the runway conditions, the wind speed and direction, the weight of the plane, the altitude

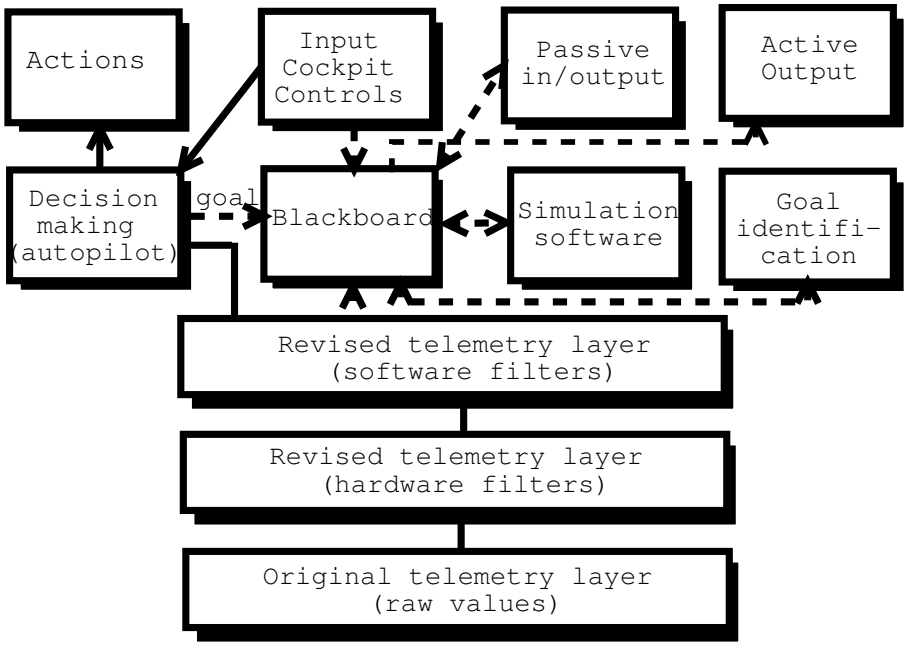


Figure 5: A Proposal architecture

of the airport and a number of other pieces of information, it is possible to make use of the list of approximations provided by Boeing. These, together with the list of corrections also, separately, provided, constitute sufficient information to estimate the distance required.

It would be preferable, however, to be able to produce an estimation of this information before landing, continually reassessing the situation. This is true not only in the case of the issue of landing but also in terms of the issue of problem detection in general. Klein et al, 1999 [5], discuss the problem detection process with particular reference to the fact that problems require data elements to be monitored for their identification — yet these data elements are not monitored in a typical situation unless the pilots are already aware that a problem may exist.

In this case, a mechanism is required that permits the plane's likely landing trajectory to be ascertained.

The time and distance covered during a landing can be approximated, very roughly, by simple geometry. The figure below shows this;

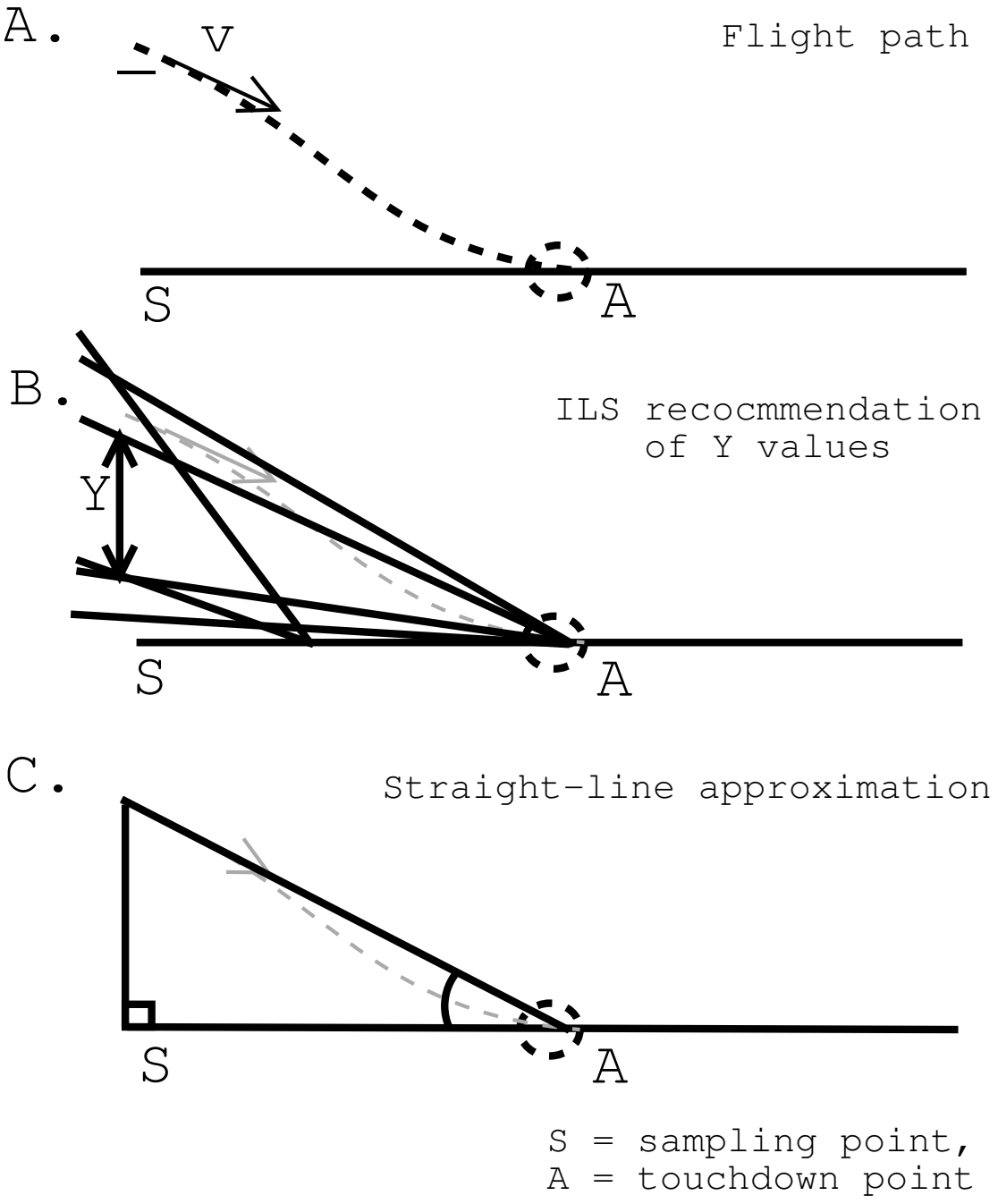


Figure 6: Formalisation of a sample landing trajectory

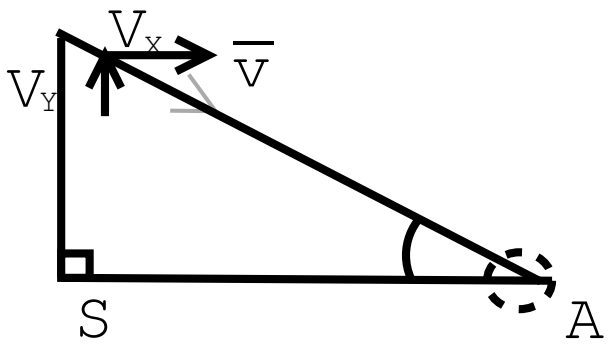
Figure A shows a sample landing trajectory. This is the real trajectory taken by the plane,

and is constantly varying based on local weather conditions and similar effects. It is hard to simulate, although a great deal of work has been put into finding equations that give a solution with no more than a small amount of error. Many of these are relatively complex, involving, for example, Monte Carlo methods of analysis.

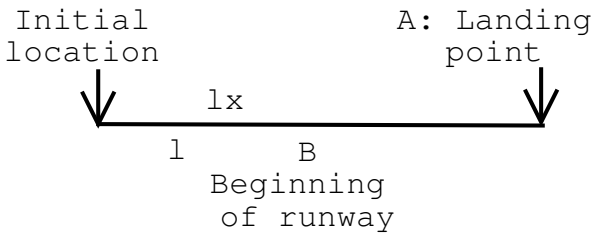
Figure B shows the ILS system, a series of beacons that provide a very simple indication to the pilot of the plane as to whether or not they are on an appropriate landing trajectory. Note that in three dimensions, the path shown by the ILS is essentially a cone, tapering towards a point. The ILS system designates the optimal path, a suboptimal path (surrounding) and incorrect/unacceptable paths.

Assuming that the aeroplane succeeds in following the tracks laid out by the ILS system, it is quite clear that the curving trajectory it describes must be completely described within the inner cone of the ILS system. Therefore, it is possible with a margin of error no greater than that permitted by the ILS system to model the landing trajectory of the aeroplane as a straight line.

This assumption, quite inaccurate but acceptable for the purpose of inclusion within this example model, permits us to describe the path mathematically, using nothing but simple geometry. This can be done as follows;



Let us label the points on the x-axis according to their relation to this problem;



According to this, it is trivially true to state the following:

$$tan\alpha = \frac{h}{x} \tag{2}$$

Therefore,

$$x = \frac{h}{\tan\alpha} = \frac{h\cos\alpha}{\sin\alpha} \tag{3}$$

It is also true that $\tan\alpha = \frac{|y|}{|x|}$.

As for the time, it is possible to note from the simple equation, or rather, definition of velocity (distance per unit time), that:

$$T(land) = T_o + \frac{x}{v} \tag{4}$$

where T(land) is the total time taken to land, and t_o is the time of origin, eg. the time at which the position of origin was taken.

Lastly, the above may be restated as follows:

$$x(land) = x_o + x = h(height) \tag{5}$$

$$x(land) = x_x - x_b \tag{6}$$

Looking at the results of this, certain show themselves to be particularly useful. If $X(land)$ is negative, the plane will land before ever reaching the runway; if $X(land) >$ runway

length, the plane will land after the runway (overshoot).

If $runway_length - X(land) <$ maximum stopping distance of the aeroplane given the runway conditions at the time, it is not possible to land using the strategy that provides the maximum stopping distance.

If the $runway_length - X(land) <$minimum stopping distance of the aeroplane given the runway conditions at the time, then the plane is unable according to this model to come to a standstill, and therefore the best advice may be to abort the landing.

17.4 Prolog implementation

The complete Prolog implementation of this program is included as an appendix to this document. As a convenience, it is also possible to consider it more mathematically - as a method of finding a solution that satisfies all of a number of provided constraints. This can be shown graphically by referring to results produced by the Prolog software, and plotted as a two-dimensional (or more, depending on the number of variables manipulated) graph of a solution space.

In order to illustrate this, consider approach to a runway at a speed of 70 m/s, in a given weather condition. The graph below shows the results of altering the brake constant (eg, the slippiness of the runway) and the distance to the runway for a given angle (so that the plane is always landing on the same trajectory, except that it is coming in later on the runway). The results are a good illustration of the solution space produced by just two variables - there are in fact literally dozens of variables that alter this in the real world, the weight of the aeroplane, the wind, humidity, angle of tilt, speed, and so on.

In effect, the spaces marked off on the figure shown below are the safe touchdown spaces; landing on the runway outside these spaces will mean that the touchdown has occurred either entirely off the runway, or excessively late!

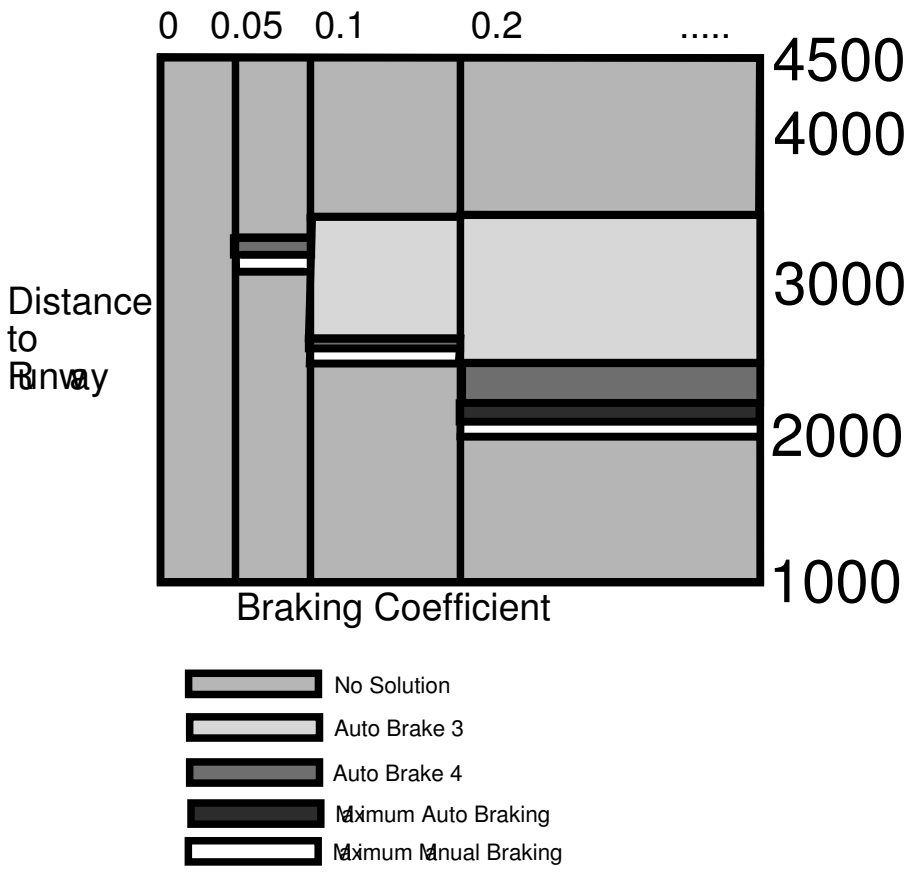


Figure 7: Sample Span of solutions for landing in a given set of conditions

17.5 Results

In order to examine what this program does more carefully, several forms of analysis may be applied. Firstly, a functional definition may be obtained simply by plugging in a range of variables and visualising the resulting information, as shown above. It is quite clear, in effect, that the software provides a method of exploring the solution space broadly defined within the Boeing databases, in such a way that much of the underlying logic is potentially transparent to the user. Thus, the knowledge needed to apply the information provided is, effectively, encoded into the software - this could be considered a first step towards crossing one of the two major barriers separating pilots from documented information, the gulf of articulation. This, of course, leaves the gulf of transfer.

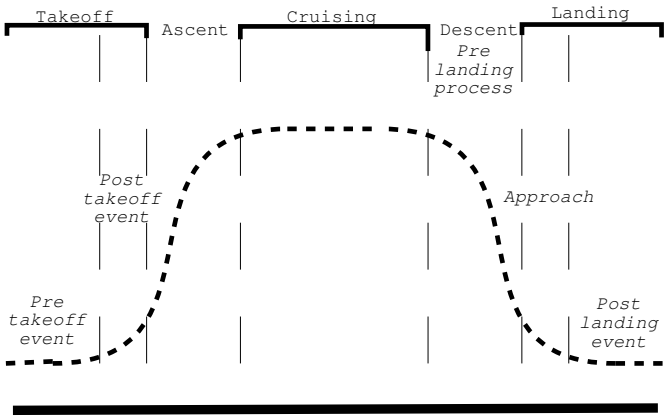


Figure 8: A simplified model of the stages in the aeroplane's fligh

17.6 Utility of this approach

One difficulty with this approach is that it tends to require an omniscient observer, in theory; which is to say that a great deal of the information, such as the precise quantity of water on the runway, is not known. To some extent, this may be alleviated by making greater use of inference, as well as considering the full potential of an agent-based system — information from any trusted source may be included into the agents' shared internal environment.

Naturally, this implies a definition of 'trusted source', as well as technically acceptable methods of ensuring physical data trustworthiness and integrity.

17.7 Context issues: a logic-based approach

Based on the model shown above, it is trivially possible to write a formal description of each of the possible stages in an aeroplane's flight, by referring to the states of the variables we have chosen to include in the environment of the machine (in other words, states which the autopilot system is able to measure/access without explicit user input). In order to do this, we start by defining the states themselves in a table as shown below;

The states that have been considered;

- Pre-landing state (descent)
- Landing (touchdown), and post-landing.
- Taxi-ing.
- Take-off (and pre-takeoff).
- Post - takeoff/climb.
- Cruising (with or without autopilot).
- Static on the ground.

Evidently there are 'emergency states' that are not explicitly covered in the descriptions of each of these states. However, emergency states cannot all be covered in this model, since by their very nature they are states in which the aeroplane has, for some reason or another, encountered a failure. Software can, however, play a useful part in the event of such an occurrence - this will be discussed later in the report.

Of these states, the variables considered of most interest are:

The wing configuration,

the directional information,

and general information such as the status of landing gear, engines, wheelbrakes, wheel revs/second, GPS, radar, fuel state and autopilot.

These items taken together, as well as a large amount of information available in the form of databases (for example, mapping information that may be cross-referenced with GPS) make up the environment within which the artificial agents operate in general. The following table is an attempt to map the various standard states listed above to the variables

available to the autopilot system;

	Taxi-ing	Pre-takeoff	Post-takeoff	Ascent	Cruising with autopilot	Cruising without autopilot	Descent	Pre-landing	Post-landing
Horizontal Accel'n	Small	+ve, large	+ve, medium	+ve	0	0 (small)	-ve	-ve, small	-ve, large
Velocity (Vx)	Small	Below critical	> Critical speed	Large	Cruise speed (constant)	Constant	>>VRef	Near VRef	< VRef
Altitude (rel. to runway)	0	0	500m	< Cruise Altitude	Cruise altitude	Cruise altitude	< Cruise Altitude	500m	0
Direction average	x and z	$y = z = 0, x = c$	$z = mx + c$	y-bell curve	straight	straight	down, as ascent	within 2 deg of straight	straight
GPS/Radar position	GPS, no radar	same	GPS,radar	GPS, radar	both	both	both	both	GPS
ILS	no	no	no	no	no	no	no	yes	no
Engine status	on	full	full	high	on	on	low	low	potentially reverse
Fuel status (rel. to start)	?	high	high	high	med	med	low	low	low
Air Brakes	off	off	off	off	off	off	controls descent	applied	fully on
Wheelbrake status	some use	off	off	off	off	off	off	off	eventually on
Landing gear	out	out	out	in	in	in	in	out	out
Autopilot	?	off	off	off	off	on	off	off	off
Wheel revs	low	increasing	0	0	0	0	0	0	high, decreases

The utility of this formal description is that it describes, given ideal circumstances, the ideal configuration of each element (although not of every single involved element, as this would be a rather longer program) in each situation. The separation into elements, eg, Pre-takeoff, post-takeoff, is done for the reason that these are the more visible physical changes in the plane's status. It is debatable whether this separation is optimal from the point of view of the user, howeve easy they are for the machine to identify.

Naturally, it is unsafe to make the assumption that the 'truth table' above is necessarily correct or entirely false (or complete — as a 'thought experiment', it certainly can lay no claim to completeness). It is precisely this that occurred in the Bangkok crash from which this essay was introduced - one element, in that case an engine, was in a state slightly non-compliant with the model, and for this reason the machine misjudged the context in which it was being used in such a way that it caused a hazardous situation to occur, or at least to worsen. Instead, the common solution to this problem may be used - a cumulative 'score' based on the number of these criteria that are correctly met. Using this, it is possible to reach a high probability of successfully defining the situation.

From this point, it is highly desirable to go a little further and attempt to gain a refined understanding of the users' goal at any one time.

17.8 Context definition

From the above, a program may be designed that is capable of taking the values of the variables mentioned above, and using them to check on the most likely context. This program is not, in fact, attempting to look for anomalies, but for 'normality', more than a semantic difference. Therefore, this particular system would fail to detect the context in novel, undescribed, or emergency situations of an unexpected nature. The usage of it is therefore as a watchdog, a first indicator of deviations from the norm. This is not to say that understanding of context in abnormal situations is not useful, since it certainly is both useful and necessary. However, it does imply that contextual information in ab-

normal situations can probably not be modelled directly to a standard model — or, more colloquially, every accident is abnormal and unexpected. The fact that this system is an 'add-on', built around the standard current cockpit, implies an interest in retaining the familiar, well-understood status quo.

In order to check the list shown previously, the following steps are taken;

A checklist is provided of the expected states of each variable for each step; the program must go through the checklist and see whether the values of the various elements match up to those expected.

If none of the answers are perfectly compliant with the state definitions provided, order them from the most to the least probable, indicating the level of confidence in each answer. Finally, and importantly, make sure that the non-compliant elements in the definitions are flagged up clearly, so that the users of the system are notified of any inconsistencies (since the non-compliance of any one variable may indicate an anomaly of which the users are unaware).

The assertion is made that the element-value pairs are 'true', and this assertion is then checked against the variables provided; if this turns out to be incorrect, the situation under scrutiny is considered to be a less likely solution. An arbitrary heuristic is used to choose the lower boundary of probability considered 'acceptable'.

This works as follows;

For each true element-value pair, $T = T + 1$ (1 is added to T - the pseudocode statement shown on the left is nonsensical in Prolog). For each false element-value pair, the element is added to an accumulator of false elements. This continues recursively until the list of element/value pairs becomes empty.

The number of correct elements is T. The number of incorrect values is $sizeof(accumulator)$, also implemented as a recursive function in Prolog (or equally, the number of incorrect values N_f is $total - T$, but either way it requires a count to be made of the list length). The degree of certainty that this simple (flawed) heuristic provides is therefore:

For each hypothesis, the degree of certainty $\sum_{n}^{(i=1)} correctness(i)$.

The percentage correctness is naturally $100 \times (\frac{correct}{n})$.

The source code for this program is available in appendix 2.

There are many suggested ameliorations to this simple method of determining context; some elements are essential to the definition of a state, for example. eg. if one is 15,000 feet from the ground level, the assumption 'post-touchdown' would be ridiculous. Of course, it is possible to include this factor rather simply - for each condition (situation) list, have a corresponding importance variable. In this way the variable's importance can be weighted in the profile, such that the failed assertion of a statement considered vital to a particular state results in a large negative score, meaning that the hypothesis is therefore discounted. A better way of showing this could be the translation of these 'vital statements' from their current position as simple additive modifiers, to factors. The resultant sum is therefore evaluated at the end of the calculation — in that sense, the logic behind the program is remarkably different. Here, the assumption is made that rather than working through the list in constant modification of a hypothesis, the system reserves judgement until the end.

In this revised model, there are 'optional' conditions, with different relative importances to the conclusion, and there are 'necessary' conditions.

'Optional' conditions, x, are not compulsory — for example, an optional condition that increases the likelihood of the 'landing' context is that the passenger indicators request that passengers remain in their seats. One of the more probable times for this to occur

is during landing; however, their absence may imply that the indicators are broken, or that the aeroplane is still at some distance from the runway. Some of these conditions are more important than others — for example, attaining a reasonable approximation reference velocity and ILS-requested trajectory is a very important part of a normal landing. The relative importances of these variables may be indicated in the optional condition modifier, α.

'Necessary' modifiers are those entirely required for a given hypothesis to be true. For example, landing gear is absolutely required for a normal landing. It is added to the model as a multiplicative factor, y.

Thus, this revised model can be summarised as shown below;

$$T = \Pi_0^m y_i \times \sum_{i=0}^{n} \alpha_i x_i \tag{7}$$

The result of running this program on a three-variable list is therefore to produce an equation of type $T = y_0 y_1 y_2 ... (\alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 ...)$.

Other changes that could usefully be made include a better or clearer definition of the place where the program receives its variables, by comparison to that briefly covered in the system architecture section. It is important to realise that the actual usefulness of this program depends entirely on the exact definitions of the values used. It is quite simple to say that one is travelling at 'cruise velocity' - or that one is at ground level, but the important issue to make clear is that ambiguity exists in this statement and must be removed for the value to be useful. For example, what is 'meant' by 'ground level'? Is it the average level of ground, or the level of ground at a specific point?

These definitions themselves are the most likely source of ambiguity in this program - the program itself literally defines nothing other than the checklist contents (the variables

used) and a trivial method for cycling through a checklist, retrieving values, and scoring them according to a simple algorithm.

Finally, as yet untouched, there is the issue of the age of data. Data beyond a certain age or data from untrustworthy sources must be identified and used with caution — for example, if a voting system implemented on a piece of the plane's hardware signals that two of the sensors disagree with the third, then it is, whilst by the definition of voting systems not sufficient reason to consider the aeroplane unsafe, nonetheless a reason to consider altering the treatment of information from those sensors. For example, if that information conflicts consistently with information from other sensors, it is necessary to come to the decision that there is an error and, once recognised, to treat it intelligently.

Should this system 'fail', which is to say, none of the existing modes suffice to explain the situation, a relatively useful method of coping with this would be to produce additional, ever- vaguer checklists that can 'catch' the problem, providing approximate but factually correct characterisations of the problem. Increasing the scope of the description given by introducing generalisations such as 'unplanned descent', or indeed 'any situation involving descent (that has not been taken up by one of the less general rules, above)' permits a certain amount of dependable information to be extracted, and from this, the relevant alerts and accompanying simulations, user manual information, and so on, may be linked to in order to ensure that the pilot is able to access the required information.

One point of some interest is that this program performs the opposite function to a successive approximation; instead, it recurses into progressively vaguer approximations until such a point as the results are deemed entirely without pattern. Thus, the software starts with a known hypothesis (a node, a perfectly, accurately described state), and is then intended to look at other precisely defined states; from then on, to proceed into ever vaguer approximation.

The intention of the software is to go from the nodes characterising a 'correct' set of values,

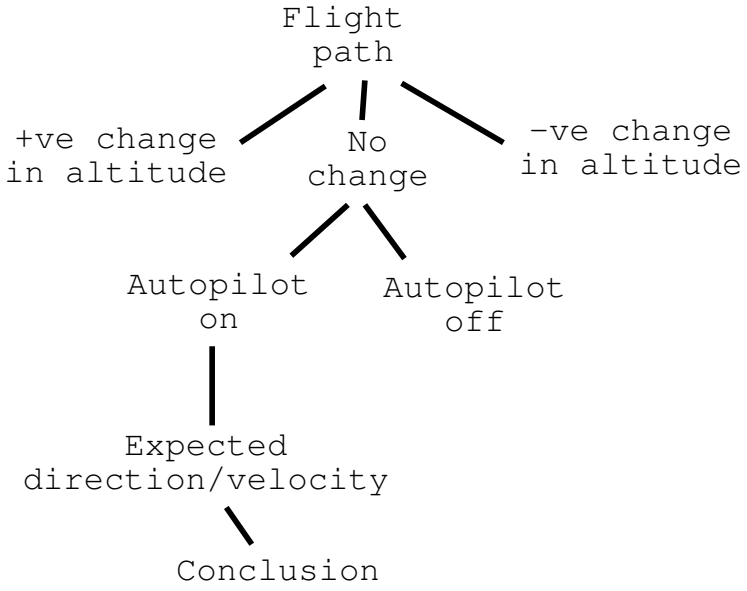


Figure 9: A partial example hypothesis

to approximation of these values.

17.9 Semantic interpretation

We have previously suggested that the situational knowledge available is dependant not only on the logic used in defining relationships between the information gleaned from the various sources, and the intention of the pilots, but also on our representation of the information itself and how the various reference variables are described. This point, a semantic issue, is really at the crux of the problem; the issue of misinterpretation or misunderstanding has been identified as an extremely significant factor in aircraft accidents[61].

It is here that the ontologies proposed by Gruber (section 12) comes into the question; the system itself must be designed to avoid ambiguity. Pilots' understanding of the various terms must be compliant — or at least, compatible — with that of the software designers and database authors.

Underlying communication is parsing, semantic interpretation, knowledge base adding to

representation of a sentence, to expand its meaning. Could a simple restricted syntax be more useful than a full English-language system? However, pilots already use a restricted language subset — a jargon designed to remove ambiguity in language. It may be that this language subset is not optimal — that is grounds for an entirely different study. Applying Occam's Razor, the issues should not be multiplied beyond necessity.

Prior domain knowledge is used to guide generalisations. Analogical reasoning maps between corresponding elements of source and target.

17.10 Treatment of temporal issues

As mentioned previously, in the discussion of agent architectures, the applicability of data of any nature may be influenced by the exact age of the data point, as well as the duration of the measurement. This is particularly true in situations where some form of inferential mathematics is performed — for example, in the case of a rainfall on a runway, exact information such as the increase in rainfall over time implies the ability to guess at the level of contamination of the runway. Similarly, information on rate of change of average windspeed may provide useful information on the probable upcoming weather.

How is this to be achieved?

The incoming telemetry is marked with a timestamp as well as a source. This permits the system to ascertain if a source stops refreshing. A little ingenuity with alternative measurements of measuring similar information will provide a manner to double - or triple - check this. Each conclusion made by an agent, including the autopilot, is also kept and provided with a timestamp for itself, as well as the timestamps of the data points on which the conclusion was based. Ideally, there is some sort of a 'decay period' for conclusions; then, a summary may occur over the various conclusions within a given time frame. This avoids the issue of transient problems with the system, a problem to which GPS in particular is remarkably prone.

18 Application of safety requirements

This discussion leads us into a revision of the safety-critical system design discussed earlier.

The system that has been proposed in these pages is not one that takes control over the aeroplane in itself, although it is designed to provide recommendations where useful. The safety implications of this system are therefore fairly minimal (since this system is an add-on which pilots are quite evidently already capable of working without, the effect of its failure, if clearly indicated, is no more than a nuisance. If the system failed in a less obvious manner, for example random corruption of data, the effects would be extremely significant. However, there are fortunately many memory storage formats that have been designed for such an eventuality that keep information on the memory contents — by checksum, for example — such that unscheduled alteration can be detected and the system declared unsafe. The archetypical example of such a system is RAID, where RAID level 1 involves 100% duplication of all system data, and RAID level 2 involves the storage of Hamming codes (checksums) for the purpose of data checking and correction, for each byte of data on the drive.

Aside from the 'safe' storage of data, there is the issue of processor or bus errors, which demands further duplication of hardware and application of error-correction techniques. Since these issues have arisen many times it is possible with a little application to locate acceptable solutions for each of the hardware aspects of the system. The worst problem is the issue of complete data security, the ability to be certain that none of the data received from the many sources is either accidentally incorrect or intentionally false — solutions exist in the domain of cryptography, but recent efforts on the part of Microsoft and others to produce a computer system immune to tampering has proven that the simplest form of security is cheap and physical, the requirement being that the system's users are unable to gain physical access to the system.

There are significant possible failure scenarios associated to the idea of a system that accepts radio input; the problem is at least as significant as the more traditional security

issues associated with computing across a network.

Fortunately, the system should perform well in terms of its integrity — part of the aim of the blackboard system is itself to increase the integrity of the traditional autopilot systems by double-checking their conclusions and the telemetry received via local instruments against data received from elsewhere as well as analysis of context and situation.

As for the traditionally required system requirements, Partridge's note that AI-based systems are difficult to engineer in the traditional manner becomes very relevant, for we have never defined what the possible capabilities of our system are — we have defined two possible tasks that it could do, and shown ways in which those tasks could be accomplished, but by the very nature of the problem there are no rigorous proofs that the systems will work — particularly in the case of the second. The same goes for system verification and validation, which rely on a clear specification of the system for checking purposes. On the other hand, it seems that the accepted wisdom of the usefulness of modular design has not been ignored, due to the use of the blackboard design. System integration and testing is certainly possible, although as yet there is very little to test other than scripts that put forward abstract responses as to the correctness or incorrectness of a given assertion, scenario or hypothesis with respect to a certain goal.

The potential faults that this system may exhibit, beyond that already mentioned (left recursivity leading to infinite loops, in the prototypical Prolog implementation), again depend entirely on the details. The exact storage and nature of the data in the knowledge base will define the level of ambiguity, as well as the structuring of the data (which will also to a great extent define the nature of the interface). Many of the potential faults depend on the interface and therefore are at this stage entirely unguessable, except in that it has already been stated that the intention is to build systems capable of providing information in a similar manner to a human agent in order to introduce the idea of the automatic systems on board the plane as, in sum, an 'entity', albeit a very restricted one, with a great deal of control over the situation.

In this case, probably the most severe failure modes are likely to stem from incorrect approaches to the interface or to the integration of such a system within an existing team, as a collaborating member rather than a simple automatic system; frustration, inappropriate antoropomorphisation, excessive expectations, or misjudgement of the system's purpose or output.

19 Teamwork and artificial agents

At the beginning of this paper, the issue of fitting the software agent into a team with human pilots was discussed briefly with reference to the curious fact that the agent-based system somewhat replaces the second officer now deemed unnecessary. That brief observation should now be examined in more detail – what place should the agent hold in a team, if it is indeed possible to find an acceptable method of interaction that permits collaborative work between human and software agents? The cockpit crew is organised into some form of a team — but is there only one acceptable organisation, and to what extent would the addition of an agent to the mix affect this?

In terms of the use of intelligent agents in team environments, two key issues have been remarked upon;

- Competence
- Trust

19.1 Competence and Trust

Competence is the question of the agent's knowledge — the question of its possesion or otherwise of the knowledge required to successfully complete its task. This element is the easier to deal with, given the nature of the agent proposed.

Trust is the question of the user's level of comfort with the agent's role.

Typically, one of the first issues that is tackled in similar research is that of establishing an identity for the agent. However, one of the issues of great importance to this design is that of trust; data integrity, for example — how is the computer to tell whether the data it uses for its decisionmaking is original to the claimed sender? What of the issue of transparency, the problem of the computer's provision of an answer with no explanation of its thought processes.

Competence implies information disclosure. Trust involves full and frank explanations of actions, facts and decisions (where required).

The question of assuring literal competence on the part of the machine, compared to that of assuring that the users of the machine feel able to trust the system. Several possible appproaches to this problem have been proposed[23]; for example :

- "Social" interface agents of the type of Microsoft "Bob"
- Anthropomorphic systems that make use of real-time dialogue, speech, gesture and gaze
- The 'computers as social actors' paradigm — agents that use humour, flattery, or self-disclosure
- Use of agents that match the user's own personality

Such systems are typically referred to as embodied conversational agents (ECAs). They are an obvious direction in which to look for the establishment of the above values — at least in theory, they are a natural avenue of research in order to deepen the trust between a user and an agent — however, in practice their benefits are found to be arguable in the face of user performance, engagement with the system, or attributions of intelligence. The design can only be an aid to "human-computer interaction, if it shows some behaviour that is functional with regard to the system's aim"[28].

Bickmore's [23] work has suggested that trust, defined as "peoples' abstract positive expectations that they can count on partners to care for them and be responsive to their needs, now and in the future", can be seen as a function of solidarity (like-mindedness) and familiarity (the increasing intimacy of a relationship). In the case that the situation permits, therefore, Bickmore shows that simulating small talk around a suitable topic can, according to an empirical study, influence users' trust in the system. The experiment used a lifesized avatar on a two-dimensional wall screen. However, the result of the work

was that there is a significant interaction between personality and trust; extroverted users responded to the system with greater trust, whilst introverts did not show this effect. Introverts and extroverts had opposite responses to the use of small talk or task-oriented discussion.

This is not the only effect; research on command and control teams [60] has shown that many of the traditional manners of exhibiting personality/a conversational or easy going nature, are unproductive in the context of times of high workload conditions or emergency. A number of other studies have suggested that application of techniques that fundamentally encourage anthropomorphism (and therefore, potentially, disappointment with the real capabilities of the system) would be ill-advised. Negroponte[51] suggests that 'All of us are quite comfortable with the idea that an all-knowing agent might live in our television set, pocket, or automobile'. However, Patrick [56] notes that whilst some researchers suggest that our years' of experience in social communication could usefully be transferred to interaction with computers, others have argued that anthropomorphism may lead to disappointment — having a human-like interface may be a short-sighted choice.

In the case of this system — an autopilot — Patrick's recommendation that developers of agent systems "only consider anthropomorphic interfaces if they truly reflect the abilities and behaviours of the agent system" is likely to be a good piece of advice. In fact, the nature of interaction in the cockpit does not lend itself at all to small-talk and social issues, since the agent representatives of the system are designed to provide expert advice. In the future, it is possible that anthropomorphisation may be a more useful technique, once some of the issues of the real-world limitations of computer systems are solved. However, at this time, there are other aspects of trust that may be more rewarding for study.

19.2 Components of trust

Trust is "the condition in which one exhibits behaviour that makes one vulnerable to someone else, not under one's control"[77]. It is this quote, cited by Patrick [56], that

makes the place of trust between the pilots and the agent utterly clear; the pilots need to rely on the agent's judgement, situational awareness, and predictive abilities. The typical difficulty between agents and users of software is the sheer distance between them; the user relies on the agent to perform an unknown and unspecified number of acts to complete a given goal. Lee, Kim & Moon [47] offer a model of trust derived specifically from e-commerce; much of that model does not apply to this problem, but one feature of it, that of the interplay between trust and perceived risk, is an interesting addition to our understanding of user acceptance. Trust itself has been described[56] as being influenced by :

- Ability to trust (on behalf of the user)
- Experience
- Predictable performance
- Comprehensive information
- Shared values
- Communication
- Interface design

...whilst perceived risk is influenced by the following :

- Risk bias
- Uncertainty
- Personal Details
- Alternatives
- Specificity
- Autonomy

Note, however, that the nature of the influence is a separate issue, and in many ways an entirely personal one; past experience with autonomous automatic systems, for example, may be negative or positive — for example, ground proximity warning systems tended to trigger too frequently at one point, leading users to ignore them entirely. Experiences of this nature may imply an automatic distrust on the user's part of all 'similar' systems, that is to say, similar from the point of view of the user, for no part of a user's likelihood to trust a system is significantly altered by a technical explanation of the system's internal functioning, unless that user is able to comprehend precisely what the system does, and in so doing, trust it because its design is technically correct.

Separation of the above lists into elements that are easily soluble in terms of interface design, system design, or process design, simplifies the issue of establishing trust into a question of tailoring these elements acceptably.

- Predictable performance implies consistency, aesthetic integrity, and perceived stability [56]
- Comprehensive information - providing information on demand
- Shared values (perhaps better stated as shared situational awareness/understanding of context)
- Communication — continual feedback from the agent [53]
- Risk management — potential solutions, alternative methods
- Autonomy, the limits of the automatic system's ability to work without human surveillance
- Interface design — there are certain visual tricks[43], such as good presentation, low brightness, cool colours, symmetrical use of colours, that have been found empirically to increase user confidence in interfaces.

However, it seems that there is little available research on the specific question of approaching the issue of trust in safety-critical situations, or indeed in professional software

in general. Yet we are aware that many aeroplane crashes of recent years have occurred at least partially due to lack of effective communication between the automatic systems and the pilots, or what Patrick refers to as uncertainty — the more that users feel they know about a system and how it operates, the less they worry about taking risks. They feel comfortable with the concept of the system's fallibility, and therefore ignore it at will.

An example of this can be found in reports of the crash of Korean Airlines flight KE801 from Seoul to Guam, at Nimitz Hill, on the approach to the runway. This area 'routinely [produces] warnings from aircraft ground proximity systems as they pass the UNZ VOR-TAC [a beacon located on the hill, used for the final approach]. Some airlines instruct their crews to expect the ground-proximity warning and to ignore it, double-checking their clearance of terrain with their aircraft's radio altimiters. The Korean Air 747 had a recent-model ground proximity warning system that provided altitude callouts to the crew before the accident.' [50]

In the logs of the flight, which terminated in a CFIT (Controlled Flight into Terrain) crash, with no obvious failure before impact, landing gear down and in an attitude consistent with landing, it was eventually discovered that the GPWS (ground proximity warning systems) had indeed been activated. The systems were audible on the cockpit voice record as warning at the 1,000 feet, 500 feet, and 100ft mark, as well as 'sink rate' and 'minimums' callouts. Yet the pilots do not appear to have responded to the warnings in any way — unremarkable in a sense, since GPWS systems have historically tended to produce a large number of false warnings, but an admirable example of the loss of trust in an automatic system — and the risk associated with it.

In the case of GPWS, an enhanced version (EGPWS) has been developed that stores a database of the world's terrain, and cross-references that information with the current location of the aircraft, providing a visual, colour-coded display of the surrounding terrain. It has also been designed to produce a longer warning period (60 seconds rather than 10), but the fact that it provides an exact visual context for its warnings is extremely impor-

tant, in that it provides users with confirmation of its warning, proof of its competence, and reason for trust (predictable, simple, comprehensible and related to the users' own situational awareness.

A further issue is that of *confirmation bias* — users who already have a particular hypothesis in mind will pick and choose from the information available to them to find elements that support that opinion.

Risk assessment when all the data is available typically shows little space for confirmation bias — there is only one conclusion, and its veracity is clear. However, when the data is incomplete (typically the case where certain variables are external to the designer's original view of a given system, for example, the potential state of the runway, the actual extent of damage to a wing, and so on), it is easy for confirmation bias to become an issue. Few enough recommendations exist for the purpose of lessening the probability of such an effect, possibly because it is rather difficult to point out clear instances of it.

Ward [78] discusses the phenomenon of confirmation bias. Methods by which confirmation bias can be controlled and minimised include making use of formalised methods of problem-solving (as, for example, the scientific method). Separation of the elements making up a decision into hypothesis; supporting elements; detracting elements, can make it difficulty to ignore evidence, and a careful statement of the problem can avoid a biased initial hypothesis.

19.3 Data security and integrity

A brief discussion of data security is, it seems, appropriate at this stage.

The assurance of data security and integrity is one that, largely due to the advent of large open networks such as the internet, has become a popular problem for research and analysis. For data integrity, solutions such as hashes and checksums have been used in

order to assure the quality of the data received as compared to that sent at the point of origin. Data security typically relies on encryption routines such as private and public key cryptography; a piece of information may be 'signed' with the digital identification of the sender, whose credentials may therefore be known beyond all reassonable doubt (PGP has been discussed in the introduction).

It is vitally important for any agent that takes information from external sources to be designed to perform all integrity checking on data, as well as to verify the indentity of the sender. For an agent to function robustly it must not only access data but also hold an inbuilt suspicion of its origins and veracity.

Many accidents occur due to conflicting information, leading to a loss of situational awareness on behalf of the pilots — this has given rise to the proposal of CRM (Salas, Fowlkes, Stout) [66], a training technique inviting pilots and copilots to feel comfortable with questioning each others' understanding of the situation or control of the aeroplane.

Even outside the possibility of intentionally sent false information, there are any number of possible causes for transmission of incorrect data - a hardware or software failure on behalf of the sender, or on behalf of the originator of the data (in a multi-agent system there are many sources of data, including the 'peers' — the other aeroplanes in the region. Thus, a further important component of data security is fault tolerance of a similar type to that discussed in classical theory :- in a variant on the theme of N-version programming, N-version data analysis is equally plausible. This is a service for which a computer is admirably well designed, since it involves a great deal of calculation. The resulting output of an N-voting N-version system is then double-checked, and in the case of anomaly (particularly repeated anomaly) it should be possible to locate the suspect source.

This does, however, require as many widely separate sources of data to be involved as possible. Otherwise, one risks allowing fundamentally the same data (eg. data extrapolated in several ways or by several different entities from the same source) to vote multiple times.

20 Teamwork

A report on team performance in Command Information Centres[44] shows that effective communication patterns, for example the use of a formalised vocabulary, limiting idle discussion, sharing information where required, may provide useful improvements. The paper showed that ommunication had a positive effect on team performance during novel (eg, unusual, previously unseen) situations, but no effect during routine situations.

The report also suggests that optimal results imply shared mental models for each teammate of the task, their teammates, and the equipment available — if all of these things are correctly understood, a minimal communication interdependancy can then exist. Team members are able to develop a common understanding of who is responsible for any given task.

Shared plans, mutual beliefs about goals and actions to be performed — and the nature of the other participants in the collaboration — are often considered necessary for successful collaboration[4].

Generally, there are two styles of team management in terms of cockpit crew; authoritarian, or democratic. Some pilots prefer to make use of one of these two styles, whilst others make use of a 'mixed' style. [6] Either style is effective in certain situations, but less advantageous in others; a democratic style of team management may lead to slower decision-making, whilst authoritarian styles of team management may lead to reluctance on the part of the co-pilot to question the pilot's actions.

These styles may have a significant effect on the breadth of shared understanding between the team members, and particularly on the potential results of a breakdown in shared understanding. Crew familiarity is a moderating variable of aviation accidents[61]; it is reasonable to imagine that part of crew familiarity is breadth of shared understanding, part of which includes the ability to communicate effectively in whatever management style the captain may choose.

A particularly extreme example of the desirability of second-guessing within teams comes from KLM flight 4805 and PanAm Boeing 1736, on March 27, 1977, when the KLM captain insisted on commencing takeoff from Tenerife, despite heavy fog, no takeoff clearance, and the knowledge that the PanAm 747 was also on the runway. Although the first and second officer were aware that they had not received clearance, and attempted to explain their concerns on the subject to the KLM captain, he insisted nonetheless on continuing the takeoff procedure, in the knowledge that longer delays might imply difficulties with lack of legal duty time. Dutch law of the time had recently begun to require time limits on flight crew activity, through a peculiar and complex calculation that the crew could not perform themselves; they had been told that taking off before a certain time would be adequate, but if that time limit was missed, it would be necessary to contact the parent company for advice.

The section of the transcript that shows this event reads :

```
KLM: KLM Radio Communications
APP: Control tower
RDO: PanAm Radio Communications
KLM-1: Pilot/Captain of KLF flight
KLM-3: Flight engineer

1705:44.8 KLM Uh, the KLM ... four eight zero five is
now ready for take-off ... uh and we're waiting for our
ATC clearance.

1705:53.4 APP KLM eight seven * zero five uh you are
cleared to the Papa Beacon climb to and maintain flight
level nine zero right turn after take-off proceed with
heading zero four zero until intercepting the three two
five radial from Las Palmas VOR.
```

1706:09.6 KLM Ah roger, sir, we're cleared to the Papa Beacon flight level nine zero, right turn out zero four zero until intercepting the three two five and we're now (at take-off).

1706:13 KLM-1 We gaan. (We're going)

1706:18.19 APP OK.

1706:19.3 RDO No .. eh.

1706:20.08 APP Stand by for take-off, I will call you.

1706:20.3 RDO And we're still taxiing down the runway, the clipper one seven three six.

1706:19.39 - 1706:23.19 RDO and APP communications caused a shrill noise in KLM cockpit - messages not heard by KLM crew.

1706:25.6 APP Roger alpha one seven three six report when runway clear

1706:29.6 RDO OK, we'll report when we're clear.
APP Thank you

1706:32.43 KLM-3 Is hij er niet af dan?
{Is he not clear then?}

```
1706:34.1  KLM-1 Wat zeg je?
{What do you say?}

1706:34.15 KLM-? Yup.

1706:34.7  KLM-3 Is hij er niet af, die Pan American?
                {Is he not clear that Pan American?}

1706:35.7  KLM-1  Jawel. {Oh yes. - emphatic}

Crash occurred at 1706:50
```

The important point is that the crew were not certain of the status of the runway — yet they accepted the word of the commanding officer. They had personal reasons to prefer a quick take-off, as they had been on duty for a long time, and were not intended to be in Tenerife at all, having been diverted there due to a bomb scare at their original destination. The captain of flight 4805 was also in a significant position of authority, as a senior pilot himself who spent much of his time training others. He did not survive the crash; the accident reports that covered the disaster held him responsible, for the three reasons that he took off without clearance, did not heed air traffic control's request that he await clearance, and did not abandon takeoff when he heard that the PanAm flight was still taxiing. Much has been made of the point that there were three officers on the flight, although the captain was the senior officer and the one in a position of authority; the other members of the cockpit crew were aware of the situation, and yet did not react.

This highlights the fact that alertness, knowledge and experience are not sufficient without the ability to work together, particularly in the event of novel or emergency situations — problem solving as a group is not well supported by establishing the principle that the senior member of the crew is to be considered to all intents and purposes infallible!

There are times when the 'authoritarian' method works well, and yet it is unhelpful in

situations where data is distributed unevenly between members of the crew and ambiguity exists, where each member of the cockpit crew is not in contact with the same information. It seems that the problem solving side of this issue could be considered an exercise in computer-supported collaborative work; the computer provides the facts, an interpretation and a framework of information, upon which the members of the team may base their collective interpretations.

Note that this particular example of a crash is another example of the potential uses for multi-agent architectures built around a central database — the authorities that control the airport fulfil this role. Since the aeroplane itself is able to 'know' where it is, there is no reason why the information cannot be stored in a central dataase and referred to by other agents. Of course, this shows up another of the difficulties with voluntary reporting of data; if an agent fails to report its location, perhaps because the system has failed, the runway would show up as empty unless backup systems were also able to update the contents of the database.

Recently, CRM — cockpit resource management — training has begun to include extensive work on the question of possible recourse in the case of the captain's error; when and if it is acceptable to take control of the plane, how to affirmatively advise pilots of the possibility that a dangerous situation may exist. A question worth asking is how an automated system comes into this — the system designed here, apart from the original avionics, does not have the ability to take charge of the plane. However, it does have the ability to report errors, and must therefore be in a similar situation to the co-pilot. The chance that a legitimate warning may be ignored for a 'bad' reason, one rooted in personality, makes it seem desirable that the dynamics of the system be further researched.

Interestingly, there are three crewmembers following the recognition of the sum of the cockpit machinery as one agent interface — one agent and two pilots. This suggests that potentially, some form of 'voting' may be potentially useful in the case of situations with multiple possible outcomes.

21 Interface issues

> "Automation that is strong, silent, and hard to direct is not a team player".
>
> – Dave Woods

This paper has demonstrated how logical inferences may be drawn from the information available to the avionics systems on an aeroplane. However, no suggestions have yet been provided as to how these may be used.

21.1 Interfaces

The avionics systems aboard an aircraft, unlike personal computers for example, are designed in order to demand as little cognitive effort as possible; it is required, particularly in these days of, as it were, interface-reductionism, that systems couch themselves behind a small, concise representation of their function. Systems are designed to approach the Holy Grail of human-centered interface design; their function and feedback are displayed clearly and simply, their input, wherever possible, is designed to be implicit on another action — the fact that produces such artifacts as the ground collision avoidance system that disactivates when approaching a runway, and the brakes that disengage themselves automatically on the activation of a motor.

To a great extent such reductionism has served the pilots well. Each instrument on an aeroplane made up of thousands would otherwise have a tiny interface of its own — data, spread across the instrument panels, willy-nilly. Grouping together such instruments' outputs, producing composite outputs, distilling information into comfortably digestable chunks, makes sense in the way that any reduction of useless distractions does. Equally, simplifying input into the vast bank of complex machinery can only work to the advantage of the pilot, who would otherwise need to spend a long time with the unrewarding task of manually operating systems that could have been controlled otherwise. Pilots have no need to be aware of every single technological servant that works for them — yet, according to Billings[24], there is a limit.

After the bulk of this report had been written, I came across a paper by Charles Billings, on the subject of human-centered design. The paper discusses the place that technology tends to be given in a human-machine interaction; he notes that a classic mistake made by designers who wish to create 'human-centered' systems is the habit of conceptualising a human-centered system, and then implementing a technology-centered system. Since the guideline 'principles' he provides are of some relevance to this paper, it is worth mentioning them here;

> First Principles of Human-Centered Systems
>
> Premise: Humans are responsible for outcomes in human-machine systems.
>
> Axiom: Humans must be in command of human-machine systems.
>
> Corollary: Humans must be actively involved in the processes undertaken by these systems.
>
> Corollary: Humans must be adequately informed of human-machine system processes.
>
> Corollary: Humans must be able to monitor the machine components of the system.
>
> Corollary: The activities of the machines must therefore be predictable.
>
> Corollary: The machines must also be able to monitor the performance of the humans.
>
> Corollary: Each intelligent agent in a human-machine system must have knowledge of the intent of the other agents.

Following the chain of thought and research that led to the production of the last hundred pages, these *First Principles* may seem somewhat familiar; it is strange, however, that these principles describe a reciprocal system — for each moment that humans are involved in the processes undertaken by a computer system, the computers are involved in the humans'; as humans may monitor the machine, the machine equally may monitor the humans. The enigma of these principles is, simply, their name.

It is certainly true that humans must be responsible for outcomes in human-machine systems, except perhaps in cases where the machine is designed to revise its own programming according to information it comes across at any time. In such cases, it is still a knotty philosophical problem to determine who, really, has learned — as Searle puts it, machines only process symbols. They do not understand. In any case, there is then the axiom; that humans must be in command of human-machine systems. Again, this is true — at least in some senses. It is this qualifier that leads to the small note of uncertainty, the fact that whilst it is undeniably true that humans are required to be in command of human-machine systems, it does not necessarily mean that humans are in fact controlling that system directly. As with all models, the grain of analysis is the trick.

An agent acting autonomously is an agent with a goal to fulfil. Even if that goal has been provided by a machine, the fact remains that at the beginning of the chain of agents is a human command. The question asked becomes, how can the events precipitated by the choice of goal be concisely and readably communicated to the user? How does the user indicate his or her state of satisfaction with the results that they monitor? How does the machine indicate its conclusions from monitoring the humans it is working with?

The name, 'Human-Centered Systems', seems to be inaccurate; it is as fairly described as team-centered systems, with a committee of intelligent agents as the junior member of the team. In a way, it is a semantic difference, but a fairly significant one. Recall that at the beginning of the paper the problem was being discussed in terms of human-computer collaboration — it seems that the solution is indeed approaching that goal.

In any case, the above provides a useful list of guidelines, which, reassuringly, is not far from that covered in this paper (it is also fairly close to two output properties of interactive systems discussed by Dubois [31], observability and honesty). Billings has not described his resons for coming to these conclusions. They bear a resemblance to the guidelines suggested for fostering trust, and to the discussion on the subject of teams — although, unfortunately, no mention is made in these principles of issues like effective

problem-solving in general and possible recourses in the case of an overly authoritarian, or preoccupied, decision-maker.

The question, of course, is how all this information — the machine's status, its opinion of the humans' performance, the actions of the machine, and the data available to it — can be presented in a usable manner. A greater bandwidth of communication may be observed given the use of a multimedia system; the amount of data required can in theory be reduced to the minimum by a thorough understanding of the humans' current situational awareness, held, for example, in dynamic user profiles [11] [15].

21.2 Volunteered information

When time is short, as is frequently true in situations where a simple database that may be queried is insufficient, it is clear that information needs to be volunteered by the system. The information should preferably be short, concise, and to the point. For example, if the system is incapable of sorting out the various inputs into a logically consistant picture of any nature, the system must ask clarification - this in the case of the Bangkok incident, perhaps. If the system believes one piece of information to be inacceptable - for example, a situation in which a landing is apparently being attempted, but all required steps have not been taken - then this information is important and should also be volunteered.

The specific dialogue of the alarm system is outside the scope of the current piece of work.

However, potential uses include; initiation of an alarm condition in the cockpit, linking to a passive interface showing detailed information on the same subject, and initiation of alarms in the passenger compartments in the event that an accident is deemed unavoidable or so probable that it is necessary to alert the passengers.

21.3 Requested information

A logic program may also act as a simple database. In this case, of course, the system is acting as a 'passive interface'. Instead of volunteering information, it provides an interface of a more traditional nature which the operators of the aeroplane may then use to examine the databases, get fluid and animated presentations of the information available and the various simulations that can be extrapolated from them, and — critically — to examine the reasoning behind the current situation, automatic settings and autopilot configuration. Ideally, information volunteered by the system described above, which is involved in interaction of such a nature that it can only produce brief, terse messages, is then represented as a new 'conversational thread' on this passive interface such that the pilots — at an appropriate time — can follow up the messages, reading through the information, its sources and the logical steps required to make the decision, to the depth required (arbitrary, but since the grain of analysis is rather fine, it is to all intents and purposes possible to go into the finest level of detail usefully available to the system as a whole).

Such a level of detail should, however, not be required. In effect, the pilots are generally assumed to be competent and sufficiently knowledgeable that very little additional information should be required to cope with any particular set of circumstances. The aims of the passive interface are two-fold:

- To ensure that the Boeing databases relevant to any current situation are quickly available, and that their contents are visible both directly and in terms of the consequences (simulation).
- To provide background information behind decisions/advice - for example, the cause of an ambiguity warning was an autopilot switch that itself operated on the information that a motor was still switched on, thus leading to an unacceptable profile (neither landing nor taking off).

In effect, it provides a way to

- Permit multithreaded 'conversation', at least retention of remaining threads (conversational segments) in the background
- Permit system queries of the blackboard system, providing full results if required, including the system's reasoning.

When one refers to multithreading in conversation, it is worth mentioning Grosz and Sidner's paper on the subject[3] ; the shifting focus of attention in discourse is represented, they suggest, by a focus stack of discourse segments, each one being a contiguous sequence of communication acts that serve some purpose. It is possible to interrupt a conversation, abandon those segments in favour of others, and later, return to earlier segments. This is the idea on which 'conversational multithreading' is based; if this is not the case, then a task interrupted by the advent of a new one is gone completely and must be rebuilt by the user of the system from the ground up.

Such logical streams of thought as item 2 are debatably useful. In the sense that they provide the pilots with a more complete representation of the reasons behind the actions/reactions of the system, they may act towards the establishment of trust between pilots and agent; a simpler approach would be the statement of initial cause and eventual effect :-

- Cause: Motor Switched on
- Effect: Brakes switched off

However, cause and effect are not necessarily evident; whilst this particular example makes a limited amount of sense, it does not provide any explanation as to why such an effect could occur based on the input that caused it. In effect, it provides a solution without explaining the problem, which is a strategy likely to result in pilots' treatment of the computer system, as before, as a 'black box' that provides output solutions for unquestioning

implementation.

The exact design of these interfaces is somewhat outside the scope of this thesis.

One proviso to be satisfied is of course a question of speed. This has been discussed previously with respect to the blackboard model; however, in general terms, it is absolutely necessary that safety-critical systems neither hang nor take an arbitrary long period of time to compute the response. For this reason, it is necessary to design the hypertext-style system specifically for the application in question, focusing on the speed as well as the reliability of the implementation.

21.4 What sort of dialogue should exist between aircraft cockpit team members?

Research by Sexton and Helmreich [61] makes a quantitative analysis of the use of language in the commercial aviation cockpit, an environment characterised in their work as 'highly technical and task-oriented'. Their statistical analysis shows a number of correlations of an interesting nature; following a caveat pointing out that language use is just one of several important factors, they proceed to note that language use of pilots vary both depending on the position of that member of the flight crew, and as a function of workload. Use of a larger number of words was positively correlated with performance, whilst it was found that flight engineers had a tendancy towards communicating using a few long words — this was negatively correlated with performance. Although it is important to remember the axiom that 'correlation does not equal causation', these studies do provide a hint as to the frequency of communication and the level of detail of its content. This provides a starting point from which to judge the acceptable frequency and content of active communication (announcements) made unbidden.

Recall that the agent system co-operative/spokesman is proposed to fill the shoes of the second officer/flight engineer, now generally considered unnecessary by the major carriers.

Therefore, limiting the approximate frequency and nature of its responses to something similar to the output of the flight engineer would appear to be a reasonable starting point.

Following the output of a piece of information on the active system, the system 'links' to the passive system shared by the pilot and first officer. This system, able to represent all of the decisions made by the autopilot, the context recognised by the agents, the additional information received from outside sources, and other data, is designed in order to represent information within a hierarchy of data — a hypermedia system.
The system, upon activation, is provided with the information that the users have been given by the active system, as well as the decision 'tree' for the announcement in question.

21.5 Discussion of hypermedia systems

Hypertext refers to the collection of nonlinear, text-based, linked nodes. Hypermedia implies that the information is in multiple formats; a collection of linked nodes of multimedia data. Potentially, this indicates that the interface is very flexible, but the price paid for this flexibility is, potentially, in clarity; unstructured data produces a confusing, difficult-to-follow interface. The typical solution to this problem is to produce interfaces which introduce clear structural elements in order that the user can follow their progress through the data. This solution is effective, but can potentially lead to negating the flexibility advantage by imposition of a potentially crippling procedurality, requiring a given pathway to be taken through the interface in order to access the data required.

The vast majority of hypermedia systems are applications of typical hypertext/HTML/HTTP systems and similar or derivative technology. It is tempting to claim with emphasis that hypertext is in no way suitable for the requirements of a safety-critical task, and it is certainly true that the typical applications of this theory are entirely unsuitable due to speed and reliability limitations. However, it is difficult to discount the usefulness of the hypermedia system in general for such an interface, if only for the reason that a hypermedia interface is the ultimate in flexibility; a loose collection of nodes that provides virtually

no innate constraints on the programmer at all.

Therefore, discussion of an interface as a hypermedia system, as a rough categorisation, is useful.

21.6 Finer-grained analysis of situated knowledge

The greatest advantage of hypermedia is the fact that, given its informal structuring between nodes, it means that almost any piece of information will lead to a further level of data — of course, there are exceptions, such as the original facts received from a measurement instrument. These facts are typically 'orphan'. This means that given a situation that requires a fine level of understanding, that level is available on demand.

21.7 Example dialogue

Here is a brief example dialogue, giving an idea of the way in which the suggested interfaces may combine to work together on a task;

In designing these, the assumption has been made that the system is, so to speak, democratic; indicating a problem issue to one of the pilots will result in the information becoming available to both. If this were not the case, then the copilot may not be the right person to provide this information to; however, a better understanding of team dynamics in the cockpit is certainly required before it is possible to tell.

```
ACTIVE: Warning; possible runway overrun

COPILOT's screen displays a graph of the landing trajectory.

PASSIVE displays weather, runway condition, trajectory,
appropriately coded in terms of colour use.
```

```
COPILOT requests runway status detail (for example, by
touching the image of the runway; verbal input would
probably be distracting)

PASSIVE displays STATUS

COPILOT requests landing strategies, fearing that conditions
will worsen in the next few minutes.

PASSIVE displays guidelines in simple text/graphical form.

COPILOT confers with PILOT, acceptable landing strategy
is chosen.

PASSIVE system displays simulation based on current
information.
```

Here is a second dialogue, that illustrates a way in which the changing capabilities of the systems could potentially change the routine of a given task; an engine fault. Whether the specific change made here (from voice communication with air traffic control to agent-based radio communication between software devices) is a good idea, is another question.

```
ACTIVE: announces engine fault

COPILOT's screen displays clear schematic of aircraft,
making it entirely clear which engine is affected.

[SYSTEM sends update to Air Traffic Control]
```

```
COPILOT: Requests information on the severity of the
malfunction.

[ATC Returns possible diversion plan to SYSTEM]

PASSIVE: Displays summary of system state.

COPILOT: Requests diversion to nearest runway

PASSIVE: Displays ATC's diversion plan
```

21.8 Potential Interfaces

Definition of an interface requires task modelling to identify it; however, based on the suggestions put forward in this paper and the information available from related projects, such as the wealth of interfaces designed to promote problem solving in the computer-supported collaborative work domain, one might venture to make a few guesses as to potentially usable solutions.

The question of explicit output directs us, by reference to the research of Michaelis and Wiggins [8], to discount information displayed graphically in favour of speech output, since, as their guidelines imply, the message to be passed is simple, short, not to be referred to at a later time, and deals with events in time — and it requires immediate attention and response.

Of course, speech output also has the advantage of attracting attention, if artificial voices are used (Schneiderman [59]), and voices can be tailored to suit the 'mood' of the task. Synthesised speech is, however, a little more difficult to understand than optimal human speech.

Michaelis & Wiggins' further conclusions support the idea that, the less urgent, more verbose, more referred-to passive dialogue would be best supported by making use of a visual display.

An example element of the interface; the screen design for choosing a hypothesis or making decisions.

Firstly, attention is drawn to the issue in question by means of a short synthesised-voice announcement; then the passive screen system is used. A leaf can be taken from the CSCW book; the information that supports a given hypothesis, and the information that refutes it, can both be displayed on screen in their relevant categories in such a way that the sum of the factors are visible simultaneously; the anomalous readings can be clearly marked, for example, by their spatial location on screen or their colour.

Similarly visual tactics can be used in the case of unfavourable predictions, for example, landing simulations showing failure; the information can be displayed graphically and linked to the suggested solutions.

Again, it must be said that under normal conditions, with a well-trained crew, barring the effects of fatigue or stress, much of the information available from this system is nothing more than confirmation of what the pilot, trained from the same manuals that feed the databases powering the system, already knows — at best, the system functions as nothing more than an *aide-memoire*.

21.9 Future amendments

The agents' environments as described here have only included a fraction of the available information. For example, no information is taken on the issues of the current location of the pilot and co-pilot, despite the fact that this could be vital knowledge[37] — if an alteration to the aeroplane's course is vitally necessary, but there are no staff available

to confirm the decision, it may be that the agent system should initiate the alteration itself.

More subtly, information such as the focus of the pilots' attention (which may be extracted looking at gaze direction, image recognition information, voice recognition software, realising of course the limitations of the setting) may be used to direct comments towards the relevant, as yet unseen features.

22 Conclusion

This report began with a discussion of a case study — a runway overrun case that occurred in bad weather. The reasons for this crash were discussed, and found to include lack of situational awareness as well as a paucity of training that covered the conditions that the pilots experienced on that day. A further point that set the scene of the accident was the communication breakdown that occurred between the pilot and copilot, as the pilot did not inform the copilot of his alteration in plans — and the communication breakdown, of equal severity, that occurred between the pilots and the autopilot which depended explicitly on given instrument settings in the cockpit for its establishment of context-awareness.

Possible solutions to each of these issues were discussed, of which there are potentially a great many, ranging from additional training – to ensure that the cockpit crew were aware of the proper action to take in the event of the poor runway conditions experienced on that occasion – to strategies that could foster a greater shared understanding of the situation. Data retrieval/mining techniques that could provide timely information for a pilot in an emergency situation are discussed, such as the linking of an active alarm system based on a computer-generated knowledge of current context, with a passive interface designed to lead the user into the available knowledge most applicable to their current situation.

The most important element of this report is that pilots understand that the various machines generally dignified with the name 'autopilot' and the associated systems (collision avoidance, agents, etc), are not simply black boxes. The definition of such systems as such is almost certainly an error, as the machines typically are quite simply in control of more information than the pilots can hope to access, as well as having finer control in certain ways than the pilots themselves can access, and a share in moment-to-moment decision making. Agents are, if not actually gifted with an ability to plan independently, provided with sufficient rules to act and formulate decisions on strong logical principles — in that sense at least they are independent decision makers and actors. Whilst anthropomorphisation is unlikely to be productive, the presentation of the computer system as a decision making member of the cockpit crew, a role that in actuality the system already holds, with

situational awareness and ability to identify errors, may have the advantage of providing pilots with a reason to consider familiarising themselves with the artificial system's own 'thought processes'.

A suitable system architecture has been identified and its usage discussed, together with its shortcomings and possible ameliorations that could be made to the system for safety reasons. Potentially useful techniques in establishing trust and effective team structure have been discussed and identified, and their uses in related situations described as case studies. Pre-existing issues in cockpit crew team structure have been discussed in view of their potential significance on the effectiveness of a central agent-based system such as the one described. The issue of three-way conversation with an essentially passive interface, as well as the question of the structure of conversations and how a hypermedia system might be used to emulate such a structure, have also been touched on.

Finally, several examples of possible dialogues between pilots, copilots and machine, have been provided.

In the end, the research contained within these pages supports the notion that a different approach to avionics systems than the current, rather utilitarian, "machine as tool" design may be advantageous. However, this notion is as yet unproven in practice. It is certainly true that given the hardware currently available and a well-designed piece of software, a number of activities that typically use much of a pilot's attention, such as monitoring instruments and the processes surrounding detection of anomalies, are transferable onto software. The final caveat remains the uniquely human issues — the fact that a computer is able to detect anomalies does not mean that the captain of the aeroplane will trust, or even put credence, on the machine's judgement. The ability of a software system to identify an erroneous decision, or request clarification, does not necessarily imply that the human agents will pay any attention to the warning – the ability to present the argument behind a given conclusion does not necessarily imply that the pilot will bow to the system's choice of logic, the presentation of the facts that provided that conclusion could as

easily lead to the pilot's use of those facts to confirm his own hypothesis.

If the experience of ground proximity warning system design has served to prove anything, it is that computers are not infallible. False-positive results serve to confuse the pilot and lessen their trust in the system, so that a further ameliorated interface was designed to permit the user to see precisely what triggered the alarm — so that the data from the sensors and the computer's predictions are displayed to the human agents in such a way that the pilot can gain an understanding of the data used by the software system. The ameliorated system is still fallible, although false positives are less frequent. However, the pilot's ability to check his mental model of the situation, so to speak, against the software's internal model, means that false alerts can be identified more reliably and easily.

There is a large class of accidents that occur, directly or indirectly, because of a breakdown in communication between members of a cabin crew. From the viewpoint proposed here, that the software collectively acts as a silent servant, it is unsurprising that communication breakdowns occur not only between human members, but also between human and software. Whilst the technical issues, and psychological/social issues such as trust, perceived competence, and data availability and relevance, can all be tackled by means of technical solutions, there remains a large group of issues such as personality clashes, team management techniques, and fatigue, that cannot be solved so simply. All that can be done in this case is to attempt to minimise the effect — but, like most problems in HCI, that, too, is a problem of multi-disciplinary and complex nature, and given the number of untested assumptions already implied in this document, such issues are a problem for another time.

```
/* Program intended to implement the Boeing databases
specific to landing in flaps 25 mode */


/* Unfortunately, Prolog's design does not lend
itself (barring extended versions) very well to
dynamic variables. The below is a simple way of
persuading Prolog to pass around a dynamic
variable, and check it against an 'enum' of
static ones. */

braking_action_description(good,X) :-
   X = good.


braking_action_description(medium,X) :-
   X = medium.



braking_action_description(poor,X) :-
   X = poor.

/* Note: Issues with esoteric units are almost
inevitable when working with Boeing's tables. */

velocity_in_metres_per_second(Velocity_In_Metres_Per_Sec,
               Velocity_In_Knots) :-
   Velocity_In_Metres_Per_Sec is 0.5144444444444 * Velocity_In_Knots.

velocity_in_knots(Velocity_In_Metres_Per_Sec,Velocity_In_Knots) :-
   Velocity_In_Knots is 1.94384449244 * Velocity_In_Metres_Per_Sec.


/* Define the possible conditions on the runway */


/* Just like a human being looking at a Boeing table,
we're going to have to do a bit of constraint
mathematics and decide where we are 'between' -
if we are flying at 140kts, then... */

/* Arbitrarily set for the time being at 3km*/

runway_length(3000).
runway(slippery).

/* Arbitrarily set flaps settings as this
is a sparsely populated database!
*/

flaps(25).


/* Reference Velocity in knots */
vref(146).

/* in tons, left unadjusted - note this is set
therefore at exactly 250 t, as far as the equations
are concerned */

plane_weight(250000).

/* per 100 feet above sea level, eg. 3 is actually
300ft */

airport_pressure_altitude(1).

/* wind, per 10kts headwind */

wind(1).

/* Provides a nice round number to give approach
speed variation per ten knots above VRef */

approach_speed_per_10kts_above_VRef(Speed,Result):-
   vref(VRef),
   Speed >= VRef,
   Result is round((Speed-VRef)/10).
```

```
approach_speed_per_10kts_above_VRef(Speed,Result):-
   vref(VRef),
   Speed < VRef,
   Result is 0.

/* Slope, Per 1 % */

slope(1).

ground_speed(X) :- ground_speed(X,Y).

ground_speed(140,140).
ground_speed(100,100).
ground_speed(60,60).

ground_speed(0,0).

ground_speed(Speed,Reference_Speed) :-
   ground_speed_(Speed,Reference_Speed).

ground_speed_(X,Y) :-
   (X > 100,
   X =< 160),
   Y is 140.

ground_speed_(X,Y) :-
   (X > 60,
   X =< 100),
   Y is 100.

ground_speed_(X,Y) :-
   (X > 0,
   X =< 60),
   Y is 60;



/* Simply a list of possible runway conditions */

runway_condition(dry).
runway_condition(damp).
runway_condition(wet).
runway_condition(puddles).
runway_condition(flooded).
runway_condition(icy).
runway_condition(wet_ice).



/* Define the braking coefficients and the
runway conditions that each implies */

/* Most of these values are from a Boeing-supplied table.

/* The following two values are not  from Boeing tables!!! */

braking_coefficient(X):-
   braking_coefficient(Y,X,Z).

braking_coefficient(puddles,0.15,0.15).
braking_coefficient(flooded,0.06,0.06).
braking_coefficient(icy,0.1,0.1).
braking_coefficient(wet_ice,0.05,0.05).
braking_coefficient(dry,0.4,0.4).
braking_coefficient(damp,0.4,0.4).
braking_coefficient(wet,0.2,0.2).


/* braking_coefficient(X,Y,Z).*/


braking_coefficient(Description,Coefficient,Reference_Coefficient) :-
   braking_coefficient_(Coefficient,Reference_Coefficient).


/* Quantization with braking coefficients */

/* We also need to be able to 'turn' braking
coefficients into 'braking_action'...*/
```

```
/* Whilst I am sure that there is an exact
description of the meaning of these items,
there is little point in implementing it
here when it is clear that this is nothing
more than an incomplete model anyway. */


braking_action(Braking_Coefficient,Braking_Action) :-
   (Braking_Coefficient >= 0,
   Braking_Coefficient =< 0.1),
   Braking_Action = poor.

braking_action(Braking_Coefficient,Braking_Action) :-
   (Braking_Coefficient > 0.1,
   Braking_Coefficient =< 0.2),
   Braking_Action = medium.


braking_action(Braking_Coefficient,Braking_Action) :-
   (Braking_Coefficient > 0.2),
   Braking_Action = good.


/*

Note: You can then reference these from within a
prolog program like so:
braking_coefficient(X,0.07,Y),braking_coefficient(X,Y,Y).

*/

braking_coefficient_(X,Y) :-
   (X > 0,
   X < 0.05),
   Y is 0.05.

braking_coefficient_(X,Y) :-
   (X > 0.05,
   X < 0.1),
   Y is 0.1.

braking_coefficient_(X,Y) :-
   (X > 0.1,
   X < 0.2),
   Y is 0.2.

braking_coefficient_(X,Y) :-
   (X > 0.2,
   X < 1),
   Y is 0.4.


runway_description(dry, 'Surface␣is␣dry').
runway_description(damp,'Surface␣shows␣a␣change␣of
        colour due to moisture').
runway_description(wet,'The␣surface␣is␣soaked,␣but
        there is no standing water').
runway_description(puddles,'Significant␣patches␣of
        standing water are visible').
runway_description(flooded,'Extensive␣standing␣water
        is visible').
runway_description(icy,'Ice␣covering').
runway_description(wet_ice,'Wet␣ice/melting␣compacted␣snow').


between(N, N1, N2):-
   (N > N1,
   N < N2) ;
   (N < N1,
   N > N2).



/* There are three forces available for
stopping the aircraft on a runway;
aerodynamic drag, reverse thrust, and
wheel braking */
```

```
/* Excitingly, the Boeing documents do
not provide any consistent view of runway
condition nomenclature, so we need to
double up a little on this one  */

/* Contribution of stopping forces to
stopping performance */

/* Ground speed 140 case

ground speed 140, contribution to total
instantaneous stopping force of (drag,
reverse thrust, brakes) in percent, total
instantaneous stopping force compared to
reference condition.

Note: This is not a flexible part of this library!
*/

/* Because of the accompanying tolerances
included above, this can be queried directly
in this way:

stopping_forces_contribution(A,B,C,D),
ground_speed(whatever),
braking_coefficient(anything).

*/

/* Dry weather */
stopping_forces_contribution(30,15,55,120) :-
   ground_speed(140),
   braking_coefficient(0.4).

/* Wet weather */
stopping_forces_contribution(20,10,70,100) :-
   ground_speed(140),
   braking_coefficient(0.2).

/* Icy weather */
stopping_forces_contribution(10,10,80,85) :-
   ground_speed(140),
   braking_coefficient(0.1).


/* Ground speed 100 case */


/* Dry weather */
stopping_forces_contribution(40,20,40,90) :-
   ground_speed(100),
   braking_coefficient(0.4).
/* Wet weather */
stopping_forces_contribution(25,20,40,70) :-
   ground_speed(100),
   braking_coefficient(0.2).
/* Icy weather */
stopping_forces_contribution(10,15,75,55) :-
   ground_speed(100),
   braking_coefficient(0.1).


/* Ground speed 60 case */

/* Dry weather */
stopping_forces_contribution(55,20,25,70) :-
   ground_speed(100),
   braking_coefficient(0.4).
/* Wet weather */
stopping_forces_contribution(35,25,40,50) :-
   ground_speed(100),
   braking_coefficient(0.2).
/* Icy weather */
stopping_forces_contribution(20,25,55,35) :-
   ground_speed(100),
   braking_coefficient(0.1).

/*
catchall 0.05 case  - no contribution
```

```
made below this point
*/

stopping_forces_contribution(0,0,0,0) :-
   ground_speed(X),
   braking_coefficient(0.05).


/*
************************************************************

        Slippery runway landing distance for flaps 25:

************************************************************

obviously, in the real world, this would be one of several
different tables.
The Boeing B747 - 400 operations manual includes versions
for each flaps setting and runway type.

However, since this implementation is really only a proof
of concept, it will stand as is for the moment.
It gives a good idea of the way that other such tables could
be integrated into the knowledge base,
and therefore there is little point to duplicating it

*/



wind_direction(headwind).
wind_direction(tailwind).

braking_configuration(Braking_action,'Auto-brake␣setting␣4',2586) :-
        flaps(25),
        braking_action_description(poor,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Maximum␣auto-brake',2580) :-
        flaps(25),
        braking_action_description(poor,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Auto-brake␣setting␣3',2586) :-
        flaps(25),
        braking_action_description(poor,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Maximum␣manual␣braking',2562) :-
        flaps(25),
        braking_action_description(poor,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Auto-brake␣setting␣3',2126) :-
        flaps(25),
        braking_action_description(medium,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Auto-brake␣setting␣4',2047) :-
        flaps(25),
        braking_action_description(medium,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Maximum␣auto-brake',2031) :-
        flaps(25),
        braking_action_description(medium,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Auto-brake␣setting␣3',1992) :-
        flaps(25),
        braking_action_description(good,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Maximum␣manual␣braking',1986) :-
        flaps(25),
        braking_action_description(medium,Braking_action),
        runway(slippery).
braking_configuration(Braking_action,'Auto-brake␣setting␣4',1751) :-
   flaps(25),
   braking_action_description(good,Braking_action),
   runway(slippery).
braking_configuration(Braking_action,'Maximum␣auto-brake',1583) :-
   flaps(25),
   braking_action_description(good,Braking_action),
   runway(slippery).
braking_configuration(Braking_action,'Maximum␣manual␣braking',1482) :-
        flaps(25),
```

```
        braking_action_description(good,Braking_action),
        runway(slippery).


/*

Landing distance adjustment; need weight per
10000 kg above 250t. for the weight one.Just
take weight and work the rest out in a function
here

*/

landing_distance_adjustment(Braking_action,weight, Weight, Adjustment):-
   braking_action_description(good,Braking_action),
   (Weight=<250000),
   runway(slippery),
   Adjustment is (Weight-250000)*55 / 10000.


landing_distance_adjustment(Braking_action,weight, Weight, Adjustment):-
   braking_action_description(medium,Braking_action),
   (Weight=<250000),
   runway(slippery),
   Adjustment is (Weight-250000)*79/10000.

landing_distance_adjustment(Braking_action,weight, Weight, Adjustment):-
   braking_action_description(poor,Braking_action),
   (Weight=<250000),
   runway(slippery),
   Adjustment is (Weight-250000)*107/10000.

landing_distance_adjustment(Braking_action,weight, Weight, Adjustment):-
   braking_action_description(good,Braking_action),
   (Weight>250000),
   runway(slippery),
   Adjustment is (Weight-250000)*55 /10000.
landing_distance_adjustment(Braking_action,weight, Weight, Adjustment):-
   braking_action_description(medium,Braking_action),
   (Weight>250000),
   runway(slippery),
   Adjustment is (Weight-250000)*79 /10000.
landing_distance_adjustment(Braking_action,weight, Weight, Adjustment):-
   braking_action_description(poor,Braking_action),
   (Weight>250000),
   runway(slippery),
   Adjustment is (Weight-250000)*107 /10000.

/* This one is per 100 ft above sea level - airport pressure altitude*/
landing_distance_adjustment(Braking_action,altitude,
         Distance_Above_Sealevel, Adjustment):-
   braking_action_description(good,Braking_action),
   runway(slippery),
   Adjustment is 64*Distance_Above_Sealevel.
landing_distance_adjustment(Braking_action,altitude,
         Distance_Above_Sealevel, Adjustment):-
   braking_action_description(medium,Braking_action),
   runway(slippery),
   Adjustment is 64* Distance_Above_Sealevel.
landing_distance_adjustment(Braking_action,altitude,
         Distance_Above_Sealevel, Adjustment):-
   braking_action_description(poor,Braking_action),
   runway(slippery),
   Adjustment is 64*Distance_Above_Sealevel.

/* Wind (per 10 kts headwind)*/
landing_distance_adjustment(Braking_action,wind, Wind, Adjustment):-
   braking_action_description(good,Braking_action),
   runway(slippery),wind_direction(headwind),
   Adjustment is -76*Wind.
landing_distance_adjustment(Braking_action,wind, Wind, Adjustment):-
   braking_action_description(medium,Braking_action),
   runway(slippery),
   wind_direction(headwind),
   Adjustment is -107*Wind.
landing_distance_adjustment(Braking_action,wind, Wind, Adjustment):-
   braking_action_description(poor,Braking_action),
   runway(slippery),
   wind_direction(headwind),
```

```
   Adjustment is -168*Wind.

landing_distance_adjustment(Braking_action,wind, Wind, Adjustment):-
   braking_action_description(good,Braking_action),
   runway(slippery),
   wind_direction(tailwind),
   Adjustment is 290*Wind.
landing_distance_adjustment(Braking_action,wind, Wind, Adjustment):-
   braking_action_description(medium,Braking_action),
   runway(slippery),
   wind_direction(tailwind),
   Adjustment is 442*Wind.
landing_distance_adjustment(Braking_action,wind, Wind, Adjustment):-
   braking_action_description(poor,Braking_action),
   runway(slippery),
   wind_direction(tailwind),
   Adjustment is 717*Wind.

/* Approach speed (Per 10 kts above VRef) VRef is the approach speed. */
landing_distance_adjustment(Braking_action,speed, Speed, Adjustment):-
   braking_action_description(good,Braking_action),
   runway(slippery),
   Adjustment is 153*Speed.
landing_distance_adjustment(Braking_action,speed, Speed, Adjustment):-
   braking_action_description(medium,Braking_action),
   runway(slippery),
   Adjustment is 183*Speed.
landing_distance_adjustment(Braking_action,speed, Speed, Adjustment):-
   braking_action_description(poor,Braking_action),
   runway(slippery),
   Adjustment is 198*Speed.

/* Slope, per 1 % slope */

landing_distance_adjustment(Braking_action,slope, Slope, Adjustment):-
   braking_action_description(good,Braking_action),
   runway(slippery),
   (Slope > 0),
   Adjustment is 46*Slope.
landing_distance_adjustment(Braking_action,slope, Slope, Adjustment):-
   braking_action_description(medium,Braking_action),
   (Slope>0),
   runway(slippery),
   Adjustment is 107*Slope.
landing_distance_adjustment(Braking_action,slope, Slope, Adjustment):-
   braking_action_description(poor,Braking_action),
   (Slope>0),
   runway(slippery),
   Adjustment is 259*Slope.


/* A simple test for aquaplaning conditions. */
/* The 210 represents tyre pressure of 210 psi. */

aquaplaning_conditions(Speed) :- (7.7 * sqrt(210))<Speed.

/* 20000 is a typical value for dry runway
conditions. Aquaplaning occurs somewhere
between 10,000 and 0*/

aquaplaning_occurring(Wheel_Torque,Speed) :-
   Wheel_Torque<10000,aquaplaning_conditions(Speed).


/* Target speed for final approach is VRef */

target_speed(149).

/*

For each knot above target speed, landing
distance increases...

*/

/*

It is now possible to make some useful
statements with all of the rules described
```

```
above

With these, we can make predictions about
landing, check out events during landing, and
offer possible solutions on-the-fly

*/


/*

For example, previous to landing, simulations
may be run as follows;


*/


/*

This just gives the angle of approach, as it were:


| *
|    *
|       *
|_____( __   <= this angle

The angle in question is typically around three
degrees, though it may be varied for obstacle
clearance purposes

*/

angle_of_approach(Angle,X_Vector_Component,Y_Vector_Component) :-
   Angle is atan( abs(X_Vector_Component) / abs(Y_Vector_Component)).

/*

This then takes the approach vector (disregarding
the third vector, Z, defined
for our purposes as 'depth' - we are assuming
that it remains constant, since otherwise it
implies that there is no sideways deviation...
This is unsafe, as are many other assumptions
made here (documented elsewhere), but for the
purposes of our demonstration it is
adequate.

Some may find the assumptions made to be vastly
inadequate, because
the problem being modelled is in no way as simple
as it is shown. This
objection could even be carried to its
philosophical extreme and restated
as disapproval of the whole principle of
modelling with simplification,
which, in essence, expert systems typically
do (granularity of model is high). Aerodynamics in
particular is far from being a trivial subject,
requiring more than a
little trigonometry to model well (and requiring
impossible amounts of
information to model perfectly). The only solution
 to this argument is
that, in effect, an informed guess
is far closer to the truth than complete
ignorance - depending on the accuracy
of the model, error margins may be
significant, but they are typically known and finite.

*/

expected_landing_distance(ExpectedDistance,Height,Angle_Of_Incidence) :-
   ExpectedDistance is Height* cos(Angle_Of_Incidence)/
               sin(Angle_Of_Incidence).
```

```
/*

 The above two equations were included for completeness. They have
the advantage of being essentially general, which may make them useful.
The below version is a mixture of the two that avoids much of the
mathematics!

*/


expected_landing_distance_LESSMATHS(X_Vector_Component,Y_Vector_Component,
                  Expected_Distance,Height) :-
   Expected_Distance is Height * abs(X_Vector_Component)/
                  abs(Y_Vector_Component) .


/*

Eg. speed at which the plane is going, and
the distance until touchdown will occur

*/


expected_landing_time(Expected_Time,Current_Time,
         X_Vector_Component,Distance) :-
   Expected_Time is Current_Time + Distance / X_Vector_Component.


/*

These are trivial to use:

expected_landing_distance_LESSMATHS(40,1,Distance,40),
expected_landing_time(Time_Until_Landing,0,40,Distance).

*/

touchdown_location(Landing_Location,
      Distance_To_Runway_From_Current_Location,
      Expected_Distance) :-
   Landing_Location is Expected_Distance -
      Distance_To_Runway_From_Current_Location.

/*

If landing location is negative, you
are actually landing before the runway!
if it is > runway length, then you will land after it,
if it is greater than runway length
minus maximum stopping distance,
you will have to apply better
braking strategies eg. reverse thrust
if it is greater than the minimum stopping
distance, then it is not possible to stop
entirely in that amount of space and it
is therefore necessary to consider a go-around.

*/


/*

Now at last it should be possible to make
one large function entitled landing_forecast,
which you may provide with current information
such as the direction vector, and it will
return a prediction as to the place where
you will land given your current strategy, and
the likely result of this

The exact chain of mathematics and logic
followed by this function are discussed elsewhere.


NB: There is too many variables handed to this
function. Ideally, they would be handled otherwise
and merely referenced here.
```

```
estimate_landing_strategy(0.5,70,2,40,59,X).

*/



estimate_landing_strategy(Braking_Coefficient,X_Vector_Component,
         Y_Vector_Component,
         Height_From_Ground,
         Distance_To_Runway,
         Recommendation) :-
   expected_landing_distance_LESSMATHS(X_Vector_Component,
               Y_Vector_Component,
               Expected_Distance,
               Height_From_Ground),
   touchdown_location(Touchdown_Location,
            Distance_To_Runway,Expected_Distance),
   velocity_in_knots(X_Vector_Component,Velocity),
   Velocity>3, /* Stall Speed */
   check_options(Braking_Coefficient,Touchdown_Location,
               Velocity,Recommendation),
   write(Recommendation).



/*

We can check for the 'cheap' errors first
to save some computational effort.

*/


check_options(Braking_coefficient,Touchdown_Location,Velocity,
                  Recommendation) :-
   (Touchdown_Location < 0),
   Recommendation = 'Current␣landing␣path␣causes␣touchdown␣too␣early',
   write(Recommendation).


check_options(Braking_coefficient,Touchdown_Location,Velocity,
                  Recommendation) :-
   runway_length(Length),
   (Touchdown_Location >= Length),
   Recommendation = 'Current␣landing␣path␣causes␣touchdown␣too␣late',
   write(Recommendation).

/*

Note:

This function only checks for the
existence of a possible landing strategy.
It doesn't check for the shortest.

Here is a typical set of values:

estimate_landing_strategy(0.3,70,2,40,59,X).


It could do, by adding the compliant solutions
one by one to a list of possible solutions
(along with the distance required to implement
them), and then sorting them.

An even simpler way of making it choose the
shortest one first is to ensure that the shortest
ones are first in the search tree! Prolog will,
unless instructed otherwise, cease on successful
search. Thus, if one places them in order of
preference (favourite to least favourite, in the
case of 'frugal' airlines, this means 'least fuel
consuming first'), the nearest-to-optimal
solution will be chosen.



 */
```

```
check_options(Braking_coefficient,Touchdown_Location,Velocity,
                  Recommendation) :-
   runway_length(Length),
   (Touchdown_Location >= 0,
   Touchdown_Location < Length),
   ground_speed(Velocity,Reference_Velocity),
   braking_coefficient(Description,Braking_coefficient,
                  Reference_coefficient),
   braking_action(Reference_coefficient,Braking_action),
   amend_distance_required(Braking_action,Velocity,
              Amendment_To_Distance_Required),
   get_required_distance(Length,Touchdown_Location,
      Amendment_To_Distance_Required,Amended_Distance_Required),
   choose_method(Braking_action,Amended_Distance_Required,Recommendation).

get_required_distance(Length,Touchdown_Location,
      Amendment_To_Distance_Required,Amended_Distance_Required):-
   Amended_Distance_Required is Length - Touchdown_Location
              + Amendment_To_Distance_Required,
   Amended_Distance_Required =< Length-Touchdown_Location.

get_required_distance(Length,Touchdown_Location,
      Amendment_To_Distance_Required,Amended_Distance_Required):-
   Temp_Length is Length-Touchdown_Location+Amendment_To_Distance_Required,
   Temp_Length =< Length,
   Amended_Distance_Required is Length-Touchdown_Location.



choose_method(Braking_action,Distance_Allowable) :-
   braking_configuration_(Braking_action,Information,Distance_Required),
   Distance_Required < Distance_Allowable.

choose_method(Braking_action,Distance_Allowable,Recommendation) :-
        braking_configuration_(Braking_action,Recommendation,
              Distance_Required),
        Distance_Required < Distance_Allowable.


braking_configuration_(Braking_action,Description,Distance_Required):-
   braking_configuration(Braking_action,Description,Distance_Required).

amend_distance_required(Braking_action,Velocity,
         Amended_Distance_Required):-
   plane_weight(Weight),
   airport_pressure_altitude(Altitude),
   wind(Wind),
   slope(Slope),
   approach_speed_per_10kts_above_VRef(Velocity,Result),
   landing_distance_adjustment(Braking_action,
      weight,Weight,Weight_Adjustment),
   landing_distance_adjustment(Braking_action,
      altitude,Altitude,Altitude_Adjustment),
   landing_distance_adjustment(Braking_action,
      wind,Wind,Wind_Adjustment),
   landing_distance_adjustment(Braking_action,
      speed,Result,Approach_Speed_Adjustment),
   landing_distance_adjustment(Braking_action,
      slope,Slope,Slope_Adjustment),
   landing_distance_adjustment(Braking_action,
      slope,Slope,Slope_Adjustment),
   Amended_Distance_Required is Weight_Adjustment
         + Altitude_Adjustment
         + Wind_Adjustment
         + Approach_Speed_Adjustment
         + Slope_Adjustment.

/*

Techniques similar to those shown above
can also be used in other places. For
example, since the system has access
to all of the telemetry available, it is
also able to make comparisons between
the real-world values and those predicted,
that is, act as a watchdog.

We have seen the plane's ability to
check for situations such as aquaplaning. It
```

```
can also note abberations such as those
that might be caused by pilot error/controls
that are misset.

This might be done as shown in context2.prolog, next.
*/
```

```
value(h_acceleration,none).
value(velocity,cruise).
value(altitude,cruise).
value(direction,straight).
value(ils,no).
value(engine,on).
/* fuel(med). - would be 'right' - let's make it wrong */
value(fuel,high).
value(airbrake,off).
value(wheelbrk,off).
value(landing_gr,in).
value(autopilot,on).
value(wheelrev,none).

checklist([h_acceleration,velocity,altitude,direction,ils,engine,
   fuel,airbrake,wheelbrk,landing_gr,autopilot,wheelrev]).

conditions([taxiing,small,small,ground,x,no,on,any,off,some,out,any,low]).

conditions([pre_takeoff,large,below_critical,ground,x,no,full,
   high,off,off,out,off,increasing]).

conditions([post_takeoff,medium,above,aboveground,any,no,full,
   high,off,off,out,off,none]).

conditions([ascent,positive,large,lessthancruise,any,no,full,
   high,off,off,in,off,none]).

conditions([cruising_with_autopilot,none,cruise,cruise,straight,
   no,on,med,off,off,in,on,none]).

conditions([cruising_without_autopilot,none,cruise,cruise,straight
   ,no,on,med,off,off,in,off,none]).

conditions([descent,negative,greater_than_reference,less_than_cruise,
   straight,no,on,ok,some_use,off,in,off,none]).

conditions([prelanding,negative,near_vref,low,straight,yes,low,
   low,applied,off,out,off,none]).

conditions([postlanding,large_negative,less_than_vref,ground,
   straight,yes,reverse,less,applied,on,out,off,high]).

countelements([],0).

countelements([Head|Tail],Count):-
   countelements(Tail,PartialSolution),
   Count is PartialSolution + 1 .


/* Then define the multiple solution bit as written
   already in the other program. */

check_hypothesis(Hypothesis,Result) :-
   checklist(Variables),
   conditions([Hypothesis|Conditions]),
   write('Hypothesis␣is␣'), write(Hypothesis),nl,
   check(Variables,Conditions,ExceptionList),
   show_exceptions(Hypothesis,Variables,ExceptionList,Result).


check_hypothesis(Hypothesis,Result) :-
   write('I␣do␣not␣know␣the␣conditions␣that␣match␣this␣hypothesis').

show_exceptions(Hypothesis,Variables,ExceptionList,Result):-
   countelements(Variables,Number_of_Variables),
   countelements(ExceptionList,Number_of_Exceptions),
   /* Ah, 2/3rds correct is not partiularly good,
      but it demonstrates the point */
        Number_of_Exceptions>=Number_of_Variables*2/3,
   DegreeOfConfidence is Number_of_Exceptions /
      Number_of_Variables * 100,
   write('To␣a␣degree␣of␣confidence␣of␣'),
   write(100-DegreeOfConfidence),
   write('%,␣the␣hypothesis␣'),
    write(Hypothesis),
   write('␣is␣incorrect.'),
   nl,
```

```
   Result = Hypothesis.

show_exceptions(Hypothesis,Variables,ExceptionList,Result):-
   countelements(Variables,Number_of_Variables),
   countelements(ExceptionList,Number_of_Exceptions),
   /* Ah, 2/3rds correct is not partiularly good,
      but it demonstrates the point */
        Number_of_Exceptions<Number_of_Variables*2/3,
   DegreeOfConfidence is 100-(Number_of_Exceptions /
      Number_of_Variables * 100),
   write('To␣a␣degree␣of␣confidence␣of␣'),
   write(DegreeOfConfidence),
   write('%,␣the␣hypothesis␣'),
    write(Hypothesis),
   write('␣is␣correct.'),
   nl,
   Result = Hypothesis.

/* There is nothing left to check - seems that we should
   return 'true' unconditionally */

check([],[],Exceptions).

/* A correct answer - no need to add any incorrect
   answers to the exception list, therefore. */

check([Variable|Variables],[Condition|Conditions],ExceptionList) :-
   value(Variable,Condition),
   check(Variables,Conditions,ExceptionList).

/* An incorrect answer - add it to the exception list! */
check([Variable|Variables],[Condition|Conditions],ExceptionList) :-
   check(Variables,Conditions,ExceptionList2),
   append(ExceptionList2,[Variable],ExceptionList).
```

References

[1] Cheltenham: Civil Aviation Authority. Joint aviation requirements 25.1309: Equipment, systems and installations. 1994.

[2] Paul De Bra, Peter Brusilovsky, and Geert-Jan Houben. Adaptive hypermedia: from systems to framework. *ACM Computing Surveys*, 31(4es):??–??, 1999.

[3] Grosz and Sidner. Attention, intentions and the structure of discourse. *Computational linguistics 12(3)*, pages 175–204, 1986.

[4] Grosz and Sidner. Plans for discourse. In *INTENTIONS and Communication*, pages 417–444. Cambridge MA MIT PRESS, 1990.

[5] G.Klein, Kawas Kaleh, and P W Baier. Zero forcing and minimum mean-square-error equalization for multiuser detection in code-division multiple-access channels. *IEEE Trans. on Veh. Technol.*, 45:276–287, 1999.

[6] R L Helmreich and H C Foushee. Why crew resource management? empirical and theoretical bases of human factors training in aviation. In *EL Weiner, BG Kanki and RL Helmreich (eds), Cockpit Resource Management p 3.45*. London: Academic Press, 1993.

[7] Draft international standard 1508. Functioety: Safety related system. *Geneva: International electrotechnical commission*, 1995.

[8] P R Michaelis and R H Wiggins. A human factors engineers' introduction to speech synthesizers. In *Directions in human-computer interaction, ed. A N BADRE and B SCHNEIDERMAN*. Norwood, New Jersey : Ablex, 1982.

[9] MoD. Safety management requirements for defence systems containing programmable electronics, part 1; rewuirements. *Draft defence standard 00-56 Issue 2*, 1995.

[10] USA: Office of the Federal Register National Archives and Records Administration. Federation aviation regulation 25,1309: Equipment, systems and installations. 1993.

[11] S. Oliviero. Human-computer interaction through natural language and hypermedia in alfresco, 1996.

[12] J Orasanu. Shared problem models and flight crew performance. In *Aviation psychology in practice*, pages 255–285. Aldersholt, England: Avebury Press, 1994.

[13] Seongbin Park. Structural properties of hypertext. In *UK Conference on Hypertext*, pages 180–187, 1998.

[14] H C Foushee R L Helmreich, J R Hackman. Evaluating flight crew performance; policy, pressures, pitfalls and promise, 1988.

[15] Oliviero Stock, Carlo Strapparava, and Massimo Zancanaro. User modelling in a multimodal dialogue system for information access.

[16] P Agre. Changing places: Contexts of awareness in computing. *Human-Computer Interaction*, 16, 2001.

[17] Xt An. Hypertext – an introduction.

[18] W Bandler and L.J. Kohout. Fuzzy power sets and fuzzy implication operators. *Fuzzy Sets and Systems*, 4:13-30, 1980.

[19] P. Bell. Using argument representations to make thinking visible for individuals and groups. *Proceedings of the Computer Support for Collaborative Learning (CSCL) CONFERENCE*, 1997.

[20] R Bell and D Reinert. Risk and system integrity concepts for safety-related control systems. *Microprocessors and microsystems*, (17):3–15.

[21] M Benerecetti, P Bouquet, and M Bonifacio. Distributed context-aware systems. *Human-Computer Interaction*, 16, 2001.

[22] J C Bezdek. Fuzzy models — what are they, and why? *IEEE Transactions on Fuzzy Systems*, 1.1:1–6, 1993.

[23] T Bickmore and J Cassell. Relational agents: A model and implementation of building user trust. *SIGCHI 01, March 31 - April 4*, 2001.

[24] C E Billings. Issues concerning human - centered intelligent systems: What's 'human-centered' and what's the problem?, 1997.

[25] Eric Bloedorn. Mining aviation safety data: A hybrid approach. *CAASD White Paper*, 2000.

[26] P Booth. *An Introduction to Human-Computer Interaction*. LEA, 1989.

[27] S Decker, S Melnik, and et al. The semantic web: Roles of xml and rdf. *IEEE Internet Computing*, September-October 2000.

[28] D M DEHN and S MULKEN. The impact of animated agents : A review of empirical research. *University of Saarland, Saarbrucken, Germany*, 1999.

[29] A K DEY, G D ABOWD, and D SALBER. A conceptual framework and a toolkit for supporting the rapid prototttyping of context-aware applications. *Human-Computer Interaction*, 16:97–166, 2001.

[30] P Dourish. Seeking a foundation for context-aware computing. *Human-computer interaction*, 16, 2001.

[31] D. Dubois and H. Prade. A class of fuzzy measures based on triangle inequalities. *Int. J. Gen. Sys.*, 8.

[32] R Engelmore and T Morgan. *Blackboard systems*. Reading, MA: Addison-Wesley, 1998.

[33] G Fischer. Articulating the task at hand and making information relevant to it. *Human-Computer Interaction*, 16:243–256, 2001.

[34] T R Gruber. Towards principles for the design of ontologies used for knowledge sharing. *Formal ontology in conceptual analysis and knowledge representation*, 1993.

[35] J. H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Reading, MA, Addison-Wesley, 1995.

[36] J Hong and J Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 2001.

[37] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards situated computing. In *In Proceedings of The First International Symposium on Wearable Computers*, pages 146–153, 1997.

[38] H Ishii and B Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. *Proceedings of CHI 97*, 1997.

[39] R. Jacob. Natural dialogue in modes other than natural language, 1992.

[40] R Jacob and et al. A software model and specification language for non-{WIM}p user interfaces. *ACM Transactions on Computer-Human Interaction*, 6(1):1–46, 1999.

[41] Robert J. K. Jacob. The future of input devices. *ACM Computing Surveys*, 28(4es):138–138, 1996.

[42] Robert J.K. Jacob, John J. Leggett, Brad A. Myers, and et al. An agenda for human-computer interaction research: Interaction styles and input/output devices.

[43] J Kim and J Y Moon. Designing emotional usability in customer interfaces: Trustworthiness of cyber-banking system interfaces. *Interacting with Computers, Vol. 10: 1-29*, 1998.

[44] G Klein. Adaptive teams. In *in Proceedings of 6th international CCRTS*, 2001.

[45] T Kletz. *Computer control and human error*. Rugby: Institute of Chemical Engineers, 1995.

[46] Doug Lea and Jos Marlowe. Interface-based protocol specification of open systems using PSL. *Lecture Notes in Computer Science*, 952:374–??, 1995.

[47] J Lee, J Kim, and J Y Moon. What makes internet users visit cyber stores again? key design factors for customer loyalty. In *in Proceedings of CHI '2000 The Hague, Amsterdam. pp. 305-312.*, 2000.

[48] N.B. Lesh, C. Rich, and C.L. Sidner. Using plan recognition in human-computer collaboration. *International Conference on User Modeling*, June 1999.

[49] P Lucas. Mobile devices and mobile data — issues of identity and reference. *Human-Computer Interaction*, 16:323–336, 2001.

[50] James T. McKenna. Guam crash underscores navaid, training concerns. *Aviation Week and Space Technology*, pages 33–35.

[51] N Negroponte. Agents: From direct manipulation to delegation. *In: J M Bradshow (ED). Software agents*, 1997.

[52] P G Neumann. *Computer-related risks*. New York: ACM press, 1995.

[53] D A Norman. *How Might People Interact with Agents, SoftwareAgents (Bradshaw, J. M.(ed.))*. AAAI Press/The MIT Press, 1997.

[54] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244, 1996.

[55] G I Parkin. Complex systems proved and improved, 1995.

[56] Andrew Patrick. Privacy, trust, agents & users: A review of human-factors issues associated with building trustworthy software agents. *Publication pending*, 2002.

[57] J M Rolfe and M F Allnut. Putting man in the picture. *New Scientist , Reprinted in ERGONOMICS AND HUMAN FACTORS 1*, pages 240–242, 16th Feb. 1967.

[58] S. A. N. Schafer, B Brumitt, and JJ Cadiz. Interaction issues in context-aware intelligent environments. *Human-Computer Interaction*, 16:363–378, 2001.

[59] B Schneiderman. *Designing the user interface: Strategies for effective human-computer interaction*. Reading, Massachusetts: Addison-Wesley, 1987.

[60] J M Schraagen and P C Rasker. Communication in command and control teams. In *in proceedings of 6th international CCRTS*, 2001.

[61] J Sexton and R L Helmreich. Analyzing cockpit communication: The links between language, performance, error and workload. *Human Performance in Extreme Environments, 5(1)*, pages 63–68, 2000.

[62] R SMITH. Shared vision. *Communications of the ACM*, December 2001.

[63] J Soler, V Julian, M Rebollo, C Carrascosa, and V Botti. Towards a real-time multi-agent system architecture. *Proceedings of the 6th International Conference on Multiagent Systems*, 2002.

[64] L Sterling and E Shapiro. *The Art of Prolog.* MIT Press, 1994.

[65] N Storey. *Safety-critical computer systems.* Addison Wesley, 1996.

[66] R J Stout, E Salas, and J E Fowlkes. Enhancing teamwork in complex environments through team training. *Group Dynamics: Theory, Research, & Practice, 1*, pages 169–182, 1997.

[67] S SU and R Bowen Loftin. A shared virtual environment for exploring and designing molecules. *Communications of the ACM*, December 2001.

[68] D Svanæs. Context-aware technology: A phenomenological perspective. *Human-Computer Interaction*, 16, 2001.

[69] H. Van Dyke Parunak, Albert D. Baker, and Steven J. Clark. The AARIA agent architecture: An example of requirements-driven agent-based system design. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 482–483, New York, 5–8, 1997. ACM Press.

[70] G. W. H. Van Es, A L C Roelen, E A C Kruijsen, and M K H Giesberts. Safety aspects of aircraft performance on wet and contaminated runways. 1998.

[71] H. Van Parunak. *Practical and Industrial Applications of Agent-Based Systems.* 1998.

[72] S Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35, 1997.

[73] S Whittaker and B O'Conaill. The role of vision in face-to-face and mediated communication. *In K.E., Finn, A.J. Sellen and S.B. Wilbur (eds.) Video-Mediated Communication.*, 1997.

[74] T Winograd. Architectures for context. *Human-Computer interaction*, 16:401–429, 2001.

[75] L Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[76] L Zadeh. The calculus of fuzzy restrictions, 1975.

[77] D E Zand. Trust and managerial problem solving. *Administrative science quarterly, 17*, pages 229 – 239.

[78] S G Ward. Decision support for what-if analysis and the confirmation bias, 2002, forthcoming.