

DEVELOPING HUMAN FACTORS CONTRACTS FOR SAFETY CRITICAL SYSTEMS

Author: Ralph Barraclough 16 September 2005

This is a report on a Project submitted for the degree of Advanced MSc in Safety Critical Systems Engineering (SCSE) in the Department of Computer Science at the University of York.

The report contains a total of 13,418 words, compiled using the MS Word word-count tool.

1.0 ABSTRACT

1.1.1 It is widely recognised that some form of human failure plays a part in the majority of industrial and transport incidents and accidents. There are a wide variety of Human Factors techniques that can be employed to evaluate human-computer interactions and the likelihood of human error. Similarly, there are a number of design and safety standards that call for the use of Human Factors techniques. The problem is, however, that there is currently no advice or systematic approach designed to integrate HF techniques into the design and safety assurance processes.

1.1.2 This paper investigates the current use of a ‘contracts’ technique developed in the software engineering field to capture the dependencies between components and modules. Recent work has extended this concept into the safety engineering field, and it is the aim of this paper to evaluate whether contracts can be used capture the often-implicit dependencies between safety-related systems and their human actors.

INDEX

<i>Paragraph</i>	<i>Description</i>	<i>Page</i>
1.0	ABSTRACT	i
1.0	INTRODUCTION	1
1.1	Motivation	1
1.2	Contracts	1
2.0	PREVIOUS WORK	2
2.1	Contract, a Definition	2
2.2	'Design By Contract'	3
2.3	Multi-Level Contracts	6
2.4	Safety Contracts in Design	8
2.5	Human Factors and Design	13
3.0	PROPOSAL - HUMAN FACTORS CONTRACTS	19
3.1	Recap of What We Gain From the Literature Search	19
3.2	Evaluating the HF Analysis Data Set	20
3.3	HF Contract	24
3.4	HF Contract Maintenance	25
4.0	CASE STUDY	26
4.1	Introduction	26
4.2	ASACS Safety Case	30
4.3	ASACS HF Contracts	31
5.0	FURTHER WORK	33
6.0	CONCLUSIONS	33
7.0	REFERENCES	34

INDEX TO FIGURES

<i>Figure</i>	<i>Title</i>	<i>Page</i>
Figure 1	- Partial Listing for 'deposit', Showing Preconditions and Postconditions	4
Figure 2	- Partial Listing for the <i>BANK_ACCOUNT</i> Class, Showing the Class Invariant	5
Figure 3	- Beugnard, Jezequel, Plouzeau and Watkins' Four Contract Levels	7
Figure 4	- Conmy, Nicholson, Purwantoro and McDermid's IMA Computing Architecture	9
Figure 5	- Example Safety Contract Showing Functional and Temporal Constraints	12
Figure 6	- Example of Class Diagram Showing Interactions and Constraints (Bates, Hawkins and McDermid 2003)	13
Figure 7	- Example HTA Decomposition	15
Figure 8	- Integrated Design and Safety 'V' Lifecycle Model	22
Figure 9	- ASACS Conceptual Diagram	27
Figure 10	- Excerpt From The ASACS Safety Case	30

INDEX TO TABLES

<i>Table</i>	<i>Caption</i>	<i>Page</i>
Table 1	- Example Contract Showing Obligations and Benefits to all Parties	2
Table 2	- Example 'deposit' Contract from the <i>BANK_ACCOUNT</i> Class	4

Table 3 - Bates, Hawkins and McDermid's Table Used to Record Class Client-Supplier Relationships and Constraints (Pre- and Post- Conditions)	12
Table 4 - Example Use Case	16
Table 5 - Summary Table Showing Where Current HF Techniques Could Be Applied In The Design Lifecycle And What They Could Contribute To The Safety Process	21
Table 6 - Table Showing Taxonomy for Early HF Process Output	23
Table 7 - Example (Fictitious) HF Contract.....	24
Table 8 - ASACS Human Factors Related Dependencies	31

1.0 INTRODUCTION

1.1 Motivation

1.1.1 Neil Storey (1996) [1] stated:

Invariably, the dangerous failure of a system will reflect a weakness in its specification, design, implementation or maintenance, and will therefore represent a human error within its development. This being the case, all dangerous failures can be seen as resulting from some sort of human error.

1.1.2 I sympathise with Storey's view above, and believe that more can be accomplished to integrate aspects of the Human Factors (HF) field more closely with the design and safety engineering effort that is dedicated towards the production of safety-related / safety-critical equipment and systems. Many of the current safety and design standards, for example IEC61508 [4], advise the use of HF considerations without explicitly advising how this can be achieved.

1.1.3 A specific area where I believe there is scope for a systematic approach to support system safety and the design process concerns dependencies between the human *actors* that operate and maintain systems and the safety of the systems themselves. It is simple to understand that system maintenance can affect system reliability and hence safety, however do the current requirements and safety engineering processes have explicit techniques for identifying and recording such dependencies?

1.2 Contracts

1.2.1 There are a number of areas of software and systems engineering that use a system of contracts to encapsulate and enforce certain necessary conditions. In this project, I aim to investigate this contracting process to see whether it is suitable or adaptable enough for use in a process to identify and capture the safety-related HF dependencies mentioned above. If this proves to be the case, then I intend to develop the process and evaluate it via a case study centred on a real system related to my work.

2.0 PREVIOUS WORK

2.1 Contract, a Definition

2.1.1 In the English language, a contract is defined as “an agreement between two or more parties, especially one that is written and enforceable by law” [1]. In general, a contract is a two-way arrangement describing the benefits that the participants are obliged to exchange, be they products, services or payment.

2.1.2 Przyblewski [3] explains that there are three types of contract: *verbal*, *implied* and *written*:

- a. *Verbal contracts*, sometimes termed the ‘gentlemen’s agreement’, are where no record of the bargain is taken; complex versions of such contracts can be difficult to enforce in law.
- b. *Implied contracts* generally concern the ‘fitness for a particular use’ of supplied goods or services. For example, a reasonable person would expect that a new kettle when filled to the mark with water, connected to a suitable electrical supply and turned on, would heat the water to boiling point; if it didn’t, then the purchaser would be entitled to suitable reparation.
- c. *Written contracts* should record clearly the details of the goods or services to be supplied, and thereby provide legally enforceable protection to both client and supplier.

2.1.3 Table 1 below depicts a further example of a contract, between a client and a car rental supplier. The table clearly shows the obligations and benefits expected by both parties.

Table 1 - Example Contract Showing Obligations and Benefits to all Parties

Party	Obligations	Benefits
Client	Provide proof of identity and valid driving license. Pay agreed rate for hire period, including third-party insurance. Return vehicle before the end of the agreed period and with full fuel tank. Pay for any damage to vehicle.	Use of agreed vehicle for agreed period of time. Vehicle has full fuel tank at collection. Vehicle is clean, maintained and relatively new.
Supplier	Provide agreed vehicle, with a full fuel tank at time of collection, for the agreed period of time. Ensure that the vehicle is relatively new, maintained in accordance with a published policy, and presented in a clean condition inside and out.	Not liable for any damages caused by the client, either to the vehicle or to any third party or property thereof. Receive agreed payment.

2.2 'Design By Contract'.

2.2.1 Bertrand Meyer is a modern day founding father of the use of the contracts concept to enhance the reliability and robustness of computer software. The notion of contracts is fundamental to Meyer's 'Eiffel' object-oriented programming language, which gives software developers a structured methodology and environment assisting in the development of reliable and reusable software components. Meyer (1992)[5] casually simplifies the aforementioned desirable properties of software quality as "the absence of bugs".

2.2.2 In the context of safety-critical (or safety-related) software, then *robustness*, i.e. the ability to respond to unexpected events in a predictable and controlled manner, is as important a property as reliability. For large and therefore complex pieces of software, it is usually impractical to carry out full-coverage dynamic testing in order to prove at run-time the reliability of all of the code's functionality. Due to this difficulty, many safety standards, for example IEC 61508 [4], allow safety arguments for such software to be based, to a degree dependent on the safety criticality of the software, upon the design and development processes employed by the developer. This approach aims to minimise the risk of systematic errors, i.e. 'bugs', being introduced during development, and to increase the likelihood of those errors being caught pre-release through specific verification techniques, e.g. peer-review and static testing.

2.2.3 Many software engineering methodologies, cites Meyer [5], extol the use of *defensive programming* techniques for the development of reliable software. Meyer, however, does not agree entirely with this, believing that the significant increase in the number of lines of code required to implement safe-guarding and error-trapping routines is perhaps as likely to introduce new errors as detect or control those that may exist within the required functionality-delivering code of the module. In preference to defensive programming, Meyer believes that the development of reliable and robust reusable software can be promoted through the use of contracts.

2.2.4 Software contracts, as implemented in the aforementioned Eiffel language, provide a method of clearly defining the obligations that both the caller (the client) and the programmer (the supplier) of a code function (or class, module, or whole application for that matter) must meet. The client and supplier obligations are expressed as *assertions* to the function; the client's obligations are known as *preconditions*, and the supplier's obligations are known as *postconditions*. Pre and postconditions are indicated in Eiffel by the keywords **require** and **ensure** respectively.

2.2.5 To illustrate this, consider an example class *BANK_ACCOUNT* that has various methods to *get balance*, *deposit*, *withdraw* etc. Table 2 below shows the client and supplier obligations and benefits for the method *deposit*, whilst Figure 1 below shows a partial listing for the code that might be written to implement *deposit*.

Table 2 - Example ‘deposit’ Contract from the *BANK_ACCOUNT* Class.

Party	Obligations	Benefits
Client	Pass a positive ¹ value as the argument.	Bank balance is increased by the amount deposited. The transaction is recorded for future reference.
Supplier	Balance is updated by adding the amount deposited to the balance at time of call. Record date and time deposit received in order to satisfy future bank statement obligations.	No action required if argument is zero or negative.

Figure 1 - Partial Listing for ‘deposit’, Showing Preconditions and Postconditions

```

Deposit (dep: CURRENCY) is
    -- Add dep to balance and record transaction date and time in deposit_list
    require
        -- precondition
        dep > 0
    do
        -- main functionality, i.e. update balance and store transaction date etc
    ensure
        -- postconditions
        balance = old balance + dep;
        num_deposits = old num_deposits + 1;
        deposit_list[num_deposits] /= Void
    end
    -- deposit

```

2.2.6 The client obligation to pass a positive sum is captured by the simple precondition ‘*dep > 0*’. The two supplier obligations are enforced by three postconditions separated by semi-colons (each semicolon equating to a logical AND in Eiffel, meaning that *each* postcondition must be satisfied). The first postcondition demonstrates the use of the **old** keyword, which allows special access to the values held in the variables of the calling object *at the time of the function call*, and may only be used in postconditions. The second and third postconditions attempt to capture the second supplier obligation by ensuring that the number of entries in the *deposit_list* has increased by one, and by ensuring that the most recent entry in the list has been connected to an appropriate object within the list, i.e. not left *Void*. (It is recognised that the second supplier obligation could be captured more strictly than this by checking the contents of the *deposit_list* in more detail, however the above should suffice to illustrate the concept).

2.2.7 In Eiffel, the satisfaction of pre- and post- conditions can be checked at run-time, depending on options that can be set when the source code is compiled. Meyer states that assertions should be used to trap systematic errors, such as a function call with a non-initialised argument, and not to handle special cases. The latter should be handled within the function

¹ In this example, withdrawals are handled by a separate method, since deposit and withdrawal transactions are recorded in different lists; hence it is illegal to effect a withdrawal via a negative deposit.

code, such as

```
if dep <= 0    -- ...do something special.
```

Meyer further concludes that “the main application of run-time assertion monitoring, then, is debugging.” Whilst this may be true in an obvious practical sense, I believe that the use of software contracts in this manner has the more fundamental advantage in that it encourages software designers and programmers to document and therefore think carefully about the requirements for each class or module. Clearly stated requirements mean that the programmer is much less likely to blunder along with his or her implementation and hence is more likely to effect the required functionality concisely and with less errors and bugs.

2.2.8 There is a third type of assertion, known as the *class invariant*, realistically only applicable in object-oriented programming. The class invariant applies throughout the life of every instance of that class, i.e. not just when certain specific methods are called, as is the case with pre- and post- conditions. Due to this all-pervasive nature, I consider that the class invariant is analogous to the generic implied contract described at paragraph 2.1.2 above. An example of a class invariant from the *BANK_ACCOUNT* example might be that the *balance* must never reduce below the *overdraft_limit*, which could be implemented as shown in the partial class declaration at Figure 2 below.

Figure 2 - Partial Listing for the *BANK_ACCOUNT* Class, Showing the Class Invariant

```
Class BANK_ACCOUNT feature  
  
    ... Attribute and method declarations ...  
  
invariant  
    -- class invariant: the bank balance must never go below the agreed overdraft limit  
    (which is stored as a positive number)  
  
    balance >= -overdraft_limit  
  
end    -- class BANK_ACCOUNT
```

2.2.9 At the beginning of this section, I described Eiffel as helping to promote reliable and reusable software. In the main, and in my opinion, this is achieved through the contract methodology’s disciplined way of recording the requirements for the software **as part of the software itself**, and not in some potentially distant Software Requirements Specification (SRS) document. With respect to code reuse, for example with component libraries, this assists greatly since the programmer can see the contract terms for the proper use of a module, i.e. the assertions, right there in black and white as part of the module declaration itself. An example of where this approach could have prevented a costly disaster was revealed by the investigation into the crash of the Ariane 5 rocket’s maiden flight on the 4th of June, 1996. Jézéquel and Bertrand (1997)[34] reveal that the crash was caused by an exception occurring within the Inertial Reference System (IRS) software; the exception was not managed (‘caught’) and so the entire software and indeed computer system crashed sealing the fate of the mission 40 seconds after take-off. The IRS software was originally written for the Ariane 4 rocket, which took the ‘horizontal bias’ parameter in 16-bit integer format. The exception occurred during a routine to convert a 64-bit integer into a 16-bit signed integer. This should only have been used to

convert numbers less than 2^{15} , however on Ariane 5 the horizontal bias was represented by a larger number and hence an overflow error occurred. Computational resources were tight when the Ariane 4 software was written, and so the conversion routine was not protected by an exception handler (also there was no need to add this protection since it was known that the horizontal bias was in 16-bit format and so couldn't cause an error). The requirements specification for this module did state, "at an obscure part of a document", that the horizontal bias should fit into 16 bits, however if this had been captured via Eiffel-like contracts then it would seem likely that the mismatch would have been noticed (assuming that the Ariane 5 software developers would have checked such things when the entire package was configured).

2.2.10 The simplicity of having the assertions embedded within the code is also beneficial with respect to the management of incremental change. If a module or class has to be amended, perhaps due to a change in the hardware with which it interfaces, then to ensure compatibility with the other code to which it is connected, the 'new' module must only ensure that the original assertions are maintained following the amendment. Put more simply, the module contract defines the external interfaces, and as long as these remain unchanged, then the rest of the world doesn't really need to know about what goes on internally to the module. This view is, of course, entirely over-simplistic, since the new implementation may well preserve the external *functional* interfaces to the module but alter the *non-functional* features, such as execution speed, which cannot readily be defined using coded assertions. A contracts-based approach can, however, still be used to capture such non-functional requirements as these; this is discussed further in the section on multi-level contracts at paragraph 2.3 below.

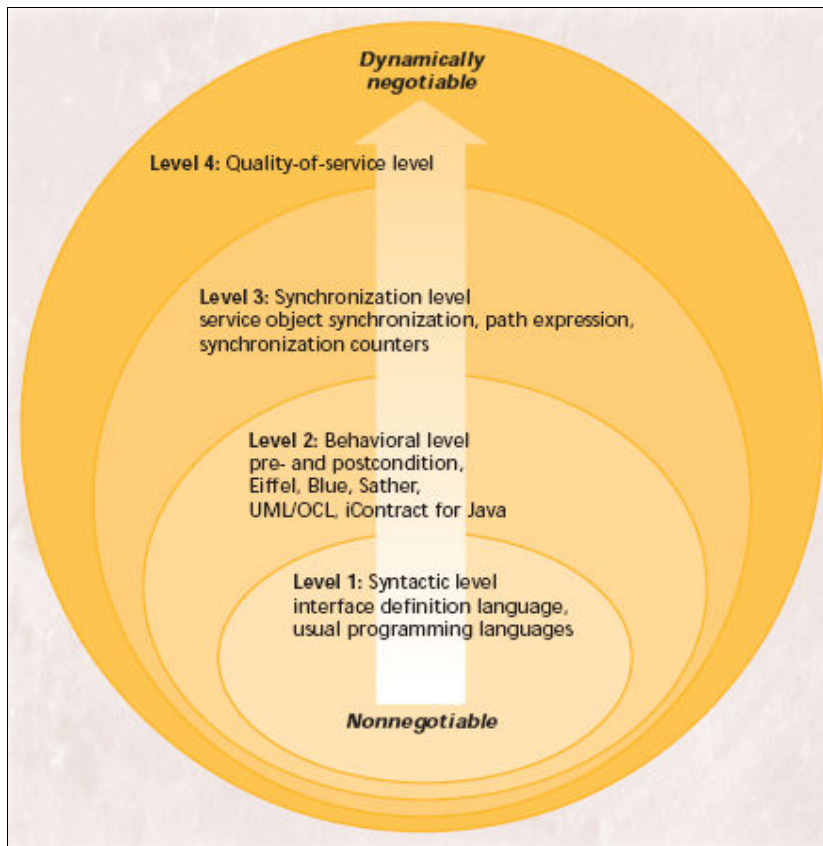
2.2.11 In a more recent interview, Meyer (Venners and Meyer 2003)[7] expanded further on the wider advantages of employing the 'Design by Contract' methodology, stating:

"In my experience, relying on these notions – that is to say, making sure when you write software that you don't just write the implementation, but also write the more abstract properties underlying the implementation in the form of contracts – provides a greatly added software development experience in several respects. It helps ensure correctness in the first place, helps debugging, helps testing, helps ensure inheritance is properly handled, helps managers, provides a quite effective form of documentation, and a few others."

2.3 Multi-Level Contracts.

2.3.1 At the beginning of this chapter, I began my introduction to the contracts concept with the explanation that there are three types of contract that we encounter in everyday life; verbal, written and implied. This idea of multiple forms of contract has been widely recognised in the software engineering domain, however here it is more usual to think of different *levels* of contract.

Figure 3 - Beugnard, Jezequel, Plouzeau and Watkins' Four Contract Levels



2.3.2 Figure 3 above shows Beugnard, Jezequel, Plouzeau and Watkins' interpretation of the *four levels* of contracts that can be applied to software components (Beugnard et al 1999 [34]). The diagram also illustrates that the contracts at the various levels can be *negotiated*, and that the degree of negotiation that can take place increases from Level 1 to Level 4. Beugnard and colleagues' paper continues to develop a methodology whereby component classes can take advantage of this negotiability of interface contracts, working towards 'contract aware' components; this is beyond the scope of this paper and is not considered further. The four levels of software contract are, however introduced below:

- a. *Level 1 or Syntactic Level Contracts* are simple object definitions, describing the interface to an object or class, such as the operations a component can perform, the input and output parameters, and the exceptions that might be raised during operation (if implemented). This type of information about a class is often referred to as the *short form* of the class definition and for many object oriented language implementations would be located in the class *Header* file (e.g. "Bank_Account.h").
- b. *Level 2 or Behavioural Level Contracts* allow the limitations on class functionality to be defined explicitly. Continuing with the example of the Bank_Account class introduced earlier, it can be seen that the short form specification does not define the class behaviour well enough for the client to be able to predict what will happen in all expected situations. For example, what will happen if the client tries to withdraw more funds than there are available, or if they attempt to deposit a negative

amount? Unless these situations are explicitly managed in the implementation using exception handling, then a technique such as the Eiffel-like pre- and post-conditions should be used to capture the level 2 contract requirements, as described at paragraph 2.2 above.

c. *Level 3 or Synchronization Level Contracts* allow the class behaviour to be defined when considering more complex cases than just the sequential operation of an isolated instantiation. The level 2 contract is sufficient so long as the methods of a component are used in isolation. However it is possible that the class may operate in a distributed environment, for example as a single server with many clients, and so we need to be able to specify how the Bank_Account objects are to behave if multiple method calls occur concurrently, e.g. what if an 'Account_Manager' client is calling a method to set a reduced overdraft limit whilst the account owner is trying to make a withdrawal up to the previous (or is it still current?) overdraft limit? Level 3 'synchronization' contracts allow the behaviour of components in this distributed environment to be specified and constrained. An example of this is the use of the Java language's 'synchronized' keyword, which forces a method to run in mutual exclusion with the other methods in the same object.

d. *Level 4 or Quality of Service Level Contracts* allow non-functional properties to be specified. Common examples of such properties are response times, data throughput rates, and computational precision. Unfortunately, it is rare for a software component to be solely responsible for the satisfaction of such quality of service requirements; third parties such as hardware components must often be relied upon, however it may not be possible to rely upon these, especially in a distributed environment where a hardware resource may be shared between several clients. Beugnard et al state, therefore, that quantified level 4 contracts can only be used in a generalized manner at present; again, this is clearly a topic of ongoing work that lies outwith the scope of this paper.

2.4 Safety Contracts in Design.

2.4.1 In the preceding sections of this chapter I have introduced the contracts concept and some instances of how it has been applied in the software engineering context. Whilst the benefits of this have been allied to safety, in that the use of contracts in software promotes reliability, robustness and re-use, the techniques described so far have not been specifically tailored to improve system safety or the safety assurance process itself.

2.4.2 The use of software and safety contracts has generated a degree of research in recent years with respect to the aerospace industry's adoption and development of Integrated Modular Avionics (IMA). This is a modern open distributed architecture for on-aircraft systems whereby a whole range of aircraft functions (navigation, engine control, centre of gravity control etc.) are implemented as software applications that are hosted on a distributed computing platform. Conmy, Nicholson, Purwantoro and McDermid (2002)[9] describe the IMA architecture shown at Figure 4 below, and the advantages to the IMA approach are as follows:

a. The applications are independent, and any interactions such as data exchange take place via the network. This modularity facilitates future upgrade of applications

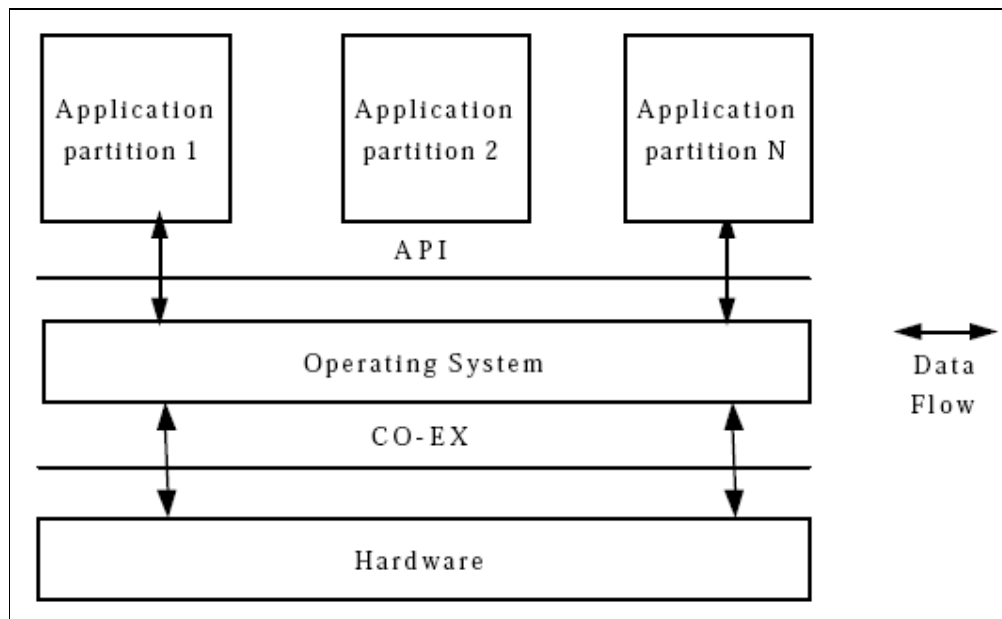
and the addition of new applications on a piecemeal basis, thereby lessening the need and risk of a 'big bang' upgrade of the entire avionics suite.

b. The applications are independent of the IMA hardware, with the Operating System (OS) acting as the buffer layer. This facilitates future expansion and upgrade of the hardware, since the applications only require guarantee of compatibility with the OS and so the provider of replacement IMA architecture only needs to ensure compatibility with the chosen OS and need not worry about any other individual requirements of the applications.

c. IMA also offers flexibility of configuration, in that the individual applications may run on different processors and occupy physically different memory each time the system starts. This also allows the ability for system reconfiguration for either limited hardware or application failure.

d. The standard platform and architecture that IMA offers enables significant cost reduction over traditional avionics systems that, even if implemented as software applications, still require discrete hardware for each sub-system. As well as these initial hardware savings, it can be seen that IMA offers further cost reductions when maintenance is considered: spares of only the one suite of hardware modules need to be held; repairs can be carried out efficiently as all modules are the same technology; and technical staff only have to be trained on a limited range of modules, again all being of like technology.

Figure 4 - Conmy, Nicholson, Purwantoro and McDermid's IMA Computing Architecture



2.4.3 The advantages introduced above that the IMA approach offers the aviation industry are straightforward enough and clearly highly desirable. However, the nature of air transport, especially where passenger aircraft are concerned, means that the aircraft and their systems need to be certified to stringent safety standards. The move towards the distributed modular IMA architecture presents specific technical problems, such as how to provide safety assurance

for the necessary memory and processor time partitioning; technical aspects such as this are considered in papers such as Conmy, Nicholson and McDermid (2003) [12], however they do not fall within the scope of this paper and are not considered further. The main focus of much of the research into IMA, however, has been towards developing a safety assurance process that is flexible enough to allow the software or hardware to be upgraded as obsolescence or other factors dictate without destroying the entire system safety assurance – this is, after all, one of the key advantages of the use of IMA. Traditional approaches have produced *safety cases*² containing argument and evidence pertaining to the *whole system*, however it can readily be seen that with this approach the entire safety process would have to be revisited if one of the IMA software applications or hardware elements was replaced. This is completely unacceptable on cost grounds alone, and therefore safety assurance processes to complement the modular flexibility of IMA are being sought.

2.4.4 From the brief introduction into IMA and the advantages described above, it can be seen that the use of contracts would be beneficial if not essential in order to achieve or maximize those advantages. In their 2003 paper “Safety Case Architecture to Complement a Contract-Based Approach to Designing Safe Systems” [11], Bates, Bate, Hawkins, Kelly and McDermid discuss the concept of generic vertical and horizontal contracts, termed ‘hierarchical’ and ‘peer’ contracts respectively:

- a. *Hierarchical Contracts* are described by Bates, Bate et al as being used to capture the dependencies between the different layers of abstraction in a system, for example a system-level hazard identification and analysis process, such as that described in IEC61508 [4], will generate derived safety requirements (DSRs) for the applications delivering the functions that could potentially lead to the identified system hazards. In turn, assuming the application is modular and implemented using OOP, the DSRs will be devolved down to the individual modules, and then down further to individual classes. I contend further that the term hierarchical contract is also applicable to those that would be used to capture those dependencies between the applications, the OS and the IMA hardware infrastructure.
- b. *Peer Contracts* encapsulate the dependencies and interactions between elements at the same system level of abstraction, for example those between the IMA applications, or those between the modules or classes within an application.

2.4.5 A ‘contract-based approach to designing safe systems’, tailored largely for the IMA sector, was proposed by Bate, Hawkins and McDermid (2003) [10]. A brief synopsis of this strategy is as follows:

- a. This approach requires the allocation of the safety requirements (DSRs) obtained during the top-level safety analysis to specific modules within the system. This generates modularity in the safety assurance that matches the nature of the overall

² A *Safety Case* is defined in the Adelard Safety Case Development Manual [13] as “a document body of evidence that provides a demonstrable and valid argument that a system is adequately safe for a given application and environment over its lifetime.”

system and thus facilitates the future upgrade process.

b. After this initial phase, an iterative process begins whereby a system model is produced clearly identifying the modules and interfaces – Bate, Hawkins and McDermid prefer to use the Unified Modelling Language (UML) [14]. Analysis of the model produces sets of both design and safety criteria, which can be used for later design decision verification, and contracts. Trade-off is an important factor in the process, as the design options will clearly not meet all of the derived assessment criteria.

c. The safety contracts are derived through analysis of the model, specifically examining *functional* and *timing* aspects of the system to identify which modules or classes and interactions can cause the system-level hazards identified in the early hazard identification and analysis phases (PHI/PHA). Techniques such as Fault Tree Analysis (FTA) are used to identify and record possible causes of system hazards; this should extend down to module and class level. State chart modelling is also used for the classes: firstly by checking to see that normal transitions don't cause hazardous behaviour; and then by mutating the state chart transitions, i.e. by analysing transitions that should not occur, thereby assessing the potential hazardous consequences of faulty behaviour. If this technique reveals potential hazardous behaviour, then the class transitions need to be constrained by appropriate contracts, i.e. by specifying Eiffel-like pre- and post- conditions on the class interactions.

d. For the analysis of the *timing* aspects, functional task sequences are analysed to identify actions that could be hazardous if performed early, late, too fast or too slow. As with the functional aspects above, the normal sequence of interactions is considered first, followed by analysis of faulty behaviour, which is modelled by adding tasks, omitting tasks and performing sequential tasks in parallel etc. This again identifies classes and activities that could contribute to system hazards and which need to be constrained by temporal pre- and post- conditions.

e. Figure 5 below is an example taken from Bate, Hawkins and McDermid (2003) showing the pre- and post- conditions derived through the described analysis process. The scenario is for a number of classes that define the behaviour of a stores management system for a fictitious military aircraft. In this case, stores are various weapons and removable fuel tanks etc. and are modelled by the Store class. Stores are attached to Stations on the fuselage and under the wings, and the system is controlled via a Stores_Manager class. The precondition requires that at least 100ms has elapsed since the release of the last store. Part of the postcondition requires that the release operation be completed within 50ms of commencement. These two temporal conditions were identified in the task analysis described above, with the quantities provided by domain experts. The remaining two requirements in the postcondition both derive from the high-level hazard analysis that identified a basic hazard that stores must not be released whilst the aircraft is on the ground. Aircraft sensors that detect when there is Weight on Wheels ('WoW') are checked via call of the Stores_Manager class method checkWoW(). From the FTA of this hazard, requirements for the Store class to both initiate a check of the WoW sensor via the Stores_Manager checkWoW() method and have its WoW variable set to false by the end of the release() operation were derived.

Figure 5 - Example Safety Contract Showing Functional and Temporal Constraints

Context Store ::release()	
pre:	previous_release.at + 100 <= Time.now
post:	WoW = false and Time.now <= Time.now@pre + 50 and Stores_Manager ^ checkWoW()

2.4.6 Bates, Hawkins and McDermid’s safety assurance strategy continues by proposing the use of tables to record the clients and suppliers for each class and the requirements and guarantees (pre- and post- conditions) needing to be satisfied for each of the client-supplier relationships. An example taken from the paper is at Table 3 below. This process must be revisited for each design iteration, but ensures that all of the interaction contracts identified in the analysis are recorded for each class involved, and not just for the class that was the focus of the analysis.

Table 3 - Bates, Hawkins and McDermid’s Table Used to Record Class Client-Supplier Relationships and Constraints (Pre- and Post- Conditions)

System : SMS_Aircraft X v.1.2			
Class : Store			
Clients	Interaction	Reqs. requiring satisfaction	Guarantees to be made
A	Release()	previous_release.at + 100 <= Time.now	WOW=false
			Time.now <= Time.now@pre + 50
			Stores_Manager ^ checkWOW ()
Supplier	Interaction	Reqs. to be met	
W	X()	pre-conditions of X()	

2.4.7 Class diagrams like Figure 6 below are used to aid the above process. The arrows indicate interaction between the classes, and the shaded rectangles denote where the class methods have pre- and post- conditions needing to be satisfied. This representation aids the assessment of design change, as follows:

- a. If, for example, a new class is introduced that calls upon existing class methods for which preconditions exist, then this is simple to spot from the diagram. Hence it is less likely that the required DSR(s) for the new class’ call upon those constrained methods will be missed.
- b. If, however, the new class is called upon by the existing system classes, then it can be seen that this might introduce new causes of system hazards since new functionality is being invoked. Thus the new interactions must be analysed as before to ensure that appropriate constraints on the new class are identified. This may, in turn,

generate new DSRs on those of the original classes making calls on the new class.

Figure 6 - Example of Class Diagram Showing Interactions and Constraints (Bates, Hawkins and McDermid 2003)



2.4.8 Changes to the system functionality, or to its context (operational role, or the operating or regulatory environment), clearly affect the whole of the system. When considering these wider changes, Bate, Hawkins and McDermid’s modular safety methodology does not address the need to revisit the hazard analysis at a system level and subsequently at lower levels to adjust or reformulate the original design and safety contracts as necessary. It does, however, offer an approach that is suited to modular architectures such as IMA and that produces safety assurance that can be updated with limited effort to take into account of ‘fit, form and function’ software or hardware upgrades.

2.5 Human Factors and Design.

2.5.1 Human Factors (HF) is defined in Interim Defence Standard 00-56 Part 1 [15] as

The systematic application of relevant information about human capabilities, limitations, characteristics, behaviours and motivation to the design of systems.

There are a number of HF techniques employed within current safety and system design processes. Some of these are specifically concerned with the design and analysis of the Human-Computer Interface (HCI), whereas others focus on human performance and how to optimise the human-machine partnership in order to achieve the organizational goals. In this section I shall introduce a number of these techniques, with the aim of identifying those that result in system or safety requirements that might be recorded appropriately via a system of HF contracts.

2.5.2 Work or Task Analysis techniques examine existing processes or work activities in order to establish the goals of the task and the interactions between the system and its users/operators. HF analysts can obtain this information via a variety of techniques, most of which involve actual users or subject matter experts (SMEs) experienced in the task or system being analysed. This ensures that the design of replacement or similar systems or processes is enriched with the views and experiences of these SMEs, which is vital, in my opinion, to the usability and hence the acceptance of the product. Various modelling and prototyping techniques are also employed further along the development cycle in order to improve product usability and acceptance.

2.5.3 Vicente (1999) [16], cited in the 2003 York University Human Factors Engineering course notes, describes a strategy of ‘Cognitive Work Analysis’, requiring five aspects of the

task to be analysed and recorded in order to provide optimal task-based information into the early design process:

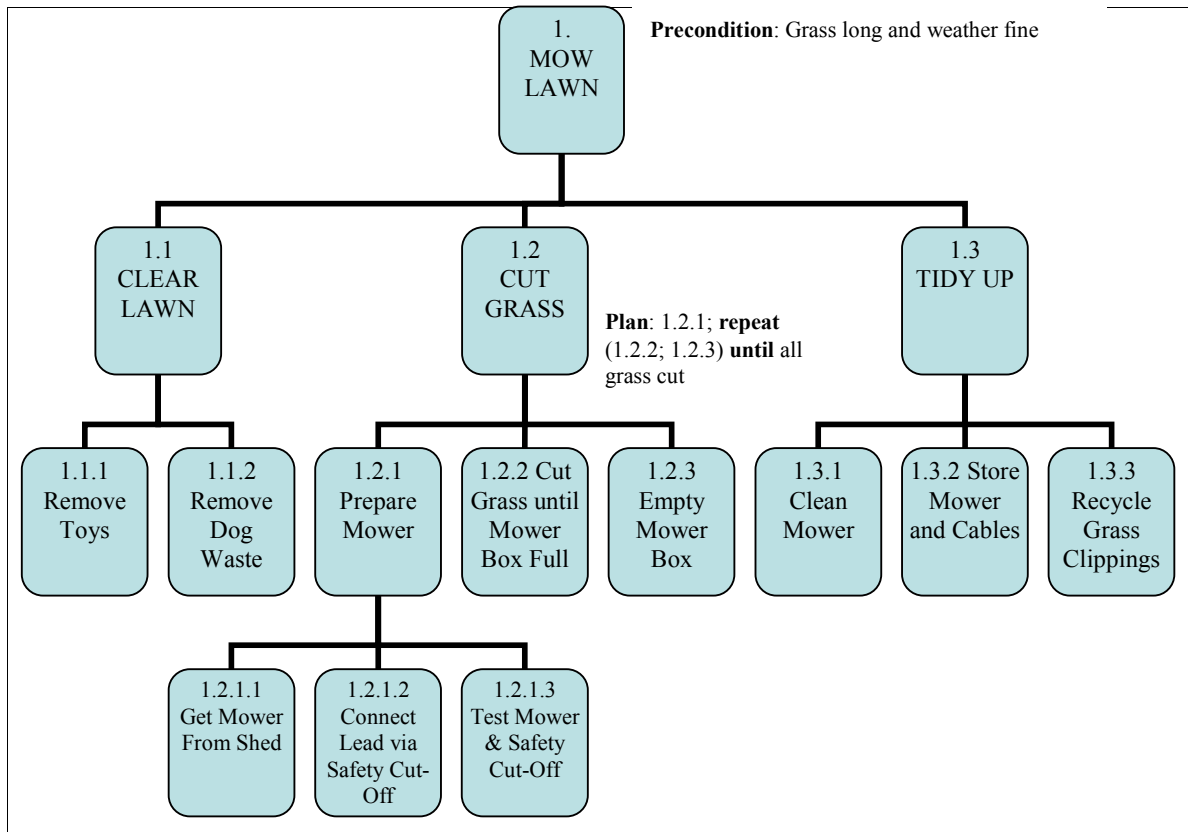
- a. The *Work Domain* must be examined in order to capture the context in which the task must be performed. It is most important that this environmental information is recorded, as factors such as the available space, the prevailing temperature and humidity etc. can constrain the range of strategies that might be available to achieve the task. It is equally important, however, that *only those environmental constraints that are fixed should be recorded*; no other details of the existing system or solution should be recorded at this point. While knowledge of the existing system's strengths and weaknesses is invaluable, it should not constrain designers into following a particular strategy or from examining alternate strategies, especially in the early design stages.
- b. The *Control Tasks* must be examined in order to capture a high-level description of the task that must be performed, again without reference to any existing or possible solutions to how it might be accomplished.
- c. High-level *Strategies* for achieving the task should be recorded in order to inform the initial design steps without prescribing too much detail. This, therefore, allows the designers the freedom to examine as wide a range of solutions as the other constraints will permit.
- d. *Social and Organisational* characteristics must also be recognised and recorded if the task or process is to be carried out within an explicit organisation or group of society. The allocation of sub-tasks must respect the hierarchy of the group otherwise the new process is likely to be rejected at some level within the group. Similarly, peer-to-peer and team-working relationships should also be recognised if the new system is intended to be used in the same way as the old (i.e. as a direct replacement). If not, then very careful and early prototyping must be employed in order to assess the likely acceptability of the various design options by the user community.
- e. *Worker Competencies* again must be explored and recorded if the new system is to be a direct replacement for an existing one. The recording of this type of information is especially important if the task or system is safety-related; the required skills that the users (and maintainers) of the new system must have will be the basis of recruiting and training policies, and will also be relied upon in the safety case.

2.5.4 A meaningful task description is required in order to help system designers understand the user requirements regarding interface design, allocation of duties (see below), and the development of user manuals and training. In order to be able to perform analysis that will lead to the production of such a meaningful task description, HF specialists need to break down the overall goal or task. This decomposition of potentially complex tasks or interactions into their constituent elements needs to be achieved in a systematic fashion; a technique known as *Hierarchical Task Analysis* (HTA³) can be used.

³ Also known as *Hierarchical Task Decomposition* (HTD)

2.5.5 Figure 7 below shows a diagrammatic example of HTA for the domestic task of mowing the lawn. The top level shows the highest-level task description, '1. Mow Lawn', with the associated preconditions for the task, i.e. that the grass actually needs to be cut and that the weather is suitable (it could be argued that I have erred with the latter precondition, in that this presupposes the solution of a human carrying out the task, however for the purposes of this example we will ignore this). The next level down shows the basic strategy comprising three main stages that I might go through in order to carry out the task to the satisfaction of my higher management. The remaining two levels of decomposition show sufficient detail to clarify and record all of the sub-tasks that are carried out in the existing solution to this task. Further decomposition might be required in order capture the fine detail that might be needed for describing non-functional requirements, such as timing, or for compiling a detailed instruction manual. Note at the second level of decomposition that a *Plan* of how to achieve sub-task 1.2 is specified. Semi-colons separate goals that are carried out in sequence, and a degree of conditional branches and loops can also be modelled: if-then, repeat-until etc.

Figure 7 - Example HTA Decomposition



2.5.6 Part of Task Analysis is Function Allocation, i.e. decomposing the goal-driven elements of the task or process into functions or units of work and allocating them to the human or machine system elements as appropriate. Examples of functions might be 'record transaction details', 'remove wheel nuts' and 'ensure reactor temperature remains within limits'. Fitts (1951) [17], again cited in the York University Human Factors Engineering course notes (2003), categorised the relative strengths and weaknesses of humans and machines with respect to performing different types of task. Clearly this is an important aspect for Function Allocation, and the following characteristics from 'Fitts' List' should be taken into account in

order to optimize the process with respect to minimising the risk of eventual human error:

- a. Humans are better at:
 - ❑ Following and improving upon *flexible* procedures.
 - ❑ Long-term information storage and the *ability to retrieve the appropriate parts when needed*.
 - ❑ Exercising judgement.
 - ❑ Recognising patterns of light and sound.
 - ❑ Inductive reasoning.

- b. Automated processor-based machines are better at:
 - ❑ Repetitively and precisely following fixed procedures (i.e. machines don't tire and won't cut corners).
 - ❑ Deductive reasoning (where the rules or decision trees are pre-programmed).
 - ❑ Monitoring control signals ad infinitum and responding to changes immediately (i.e. machines don't get bored and lose concentration).
 - ❑ The precise and controlled application of force, e.g. the control of milling machines.

2.5.7 The HTA example described at paragraph 2.5.5 above is somewhat simplistic in that it does not describe a system with many interacting components. In reality, many modern systems, such as an online bookstore, offer complex functionality and interact with many different human and automated *actors* in order to achieve the system and user goals. Whilst HTA is still useful for initial task decomposition, a more powerful technique to capture and model these numerous complex interactions is required; this is provided by the *Use Case* methodology as described by Cockburn (2001)[18].

Table 4 - Example Use Case

Use Case:	5.14
Actors:	User, Customer Database.
Preconditions:	1. User signed-in to 'ouse.com' site. 2. Shopping Basket not empty (this is implied, as 'Proceed to Check-Out' option would not be available otherwise).
Rationale:	To capture interactions between the user and the system during 'Checking Out'.
Main Success Scenario:	
Trigger Event:	User selects 'Proceed to Check-Out' from main home page.
Basic Flow of Events:	
	1. System retrieves all data from user's Shopping Basket record.
	2. System displays, for each book within Shopping Basket: title, author, required quantity, and current price.
	...
	8. System displays user's name and address, and displays 'Proceed With Purchase' and 'Cancel' buttons.
	...
	12. System calls <u>Pay By Credit Card</u> use case.
	13. System returns to main home page.

Successful Post Conditions:	<ol style="list-style-type: none"> 1. User still signed-in to 'ouse.com' site. 2. Display shows main home page.
Secondary Scenario: 6a. User Elects Not Pay By Credit Card.	
At step 11, user responds negatively.	
11.1 System asks if user wishes to pay by cheque, informing that the books cannot be dispatched and/or orders be made until the cheque has cleared.	
11.2 User responds positively.	
11.3 System calls <u>Pay By Cheque</u> use case.	
11.4 System returns to main home page.	
Post Conditions:	<ol style="list-style-type: none"> 1. User still signed-in to 'ouse.com' site. 2. Display shows main home page.
Secondary Scenario: 6b. User Elects Not Pay By Cheque...	

2.5.8 Table 4 above (Barraclough 2002 [19]) is an excerpt of a use case and depicts the majority of the characteristics of a task that can be captured using this method. It can be seen that scenario assumptions are captured via preconditions, and that the required end-states are likewise captured as postconditions. The *Main Success Scenario* records the sequence of interactions between the involved actors, and *Secondary Scenarios* and *Exception Scenarios* (not shown) are employed to record alternative paths. To adequately describe the functionality and interactions of a system such as the 'ouse.com' bookstore, a related family of use cases is compiled. Each significant task should be recorded as a use case in its own right, however it might require more than one use case per task if it is impractical to capture all of the relevant scenarios for the task in a single use case using secondary scenarios.

2.5.9 UML recognises use cases and offers a *Use Case Diagram*, which can give a simple graphical representation of the main elements of the system, the 'actors', and their interaction dependencies. The Primary Actor is generally the user, however it could be an internal system actor such as a database, or a machine abstraction of a hardware device, such as an I/O port, if that is the focus of the analysis. Cockburn states that use case diagrams should only be used as an index to the main use cases, as the diagram is only able to show the tasks and the actors but no useful detail concerning their interaction.

2.5.10 A number of techniques are available that attempt to assign probabilistic metrics to human task performance, or more to the point to the likelihood of human error. As a safety engineer this is very desirable, as most accidents are attributed to human error of some sort, and so this would allow human fallibility to be modelled alongside technical components using techniques such as Fault Tree Analysis (FTA). I, like many more learned HF specialists, are sceptical of such techniques and would not rely on such numerics due to the complexities of human cognitive abilities. Leveson (1995)[20] casts doubt over the use of *Human Error Probabilities (HEPs)* stating

Using historical data from a large number of systems has two main drawbacks: (1) the data and tasks from one system may not apply to a different system, and (2) data collection is often biased, incomplete, or inaccurate.

In addition, we all respond differently to the scenarios presented to us, and to the many other *performance shaping factors / error producing conditions* (e.g. environmental factors and emotional stresses) that techniques such as Human Error Assessment and Reduction Technique (Williams 1986 cited in the University of York Human Factors Engineering course notes

2002/3) aim to encapsulate. If such quantitative techniques are used in the safety engineering process, then I believe that it should be done in a conservative manner. Pessimistic estimates of human failure probabilities have been used by the MOD (Royal Air Force ASACS Safety Case 2005 [21] and others) via the use of Event Tree Analysis in order to derive coarse system safety targets for equipment in the Air Traffic Management environment. This approach has been approved by an Independent Safety Auditor, and has resulted in pragmatic equipment safety targets being set by avoiding the allocation of the entire risk budget onto the equipment, which would have led to harsher targets and thus over-engineering of the product. Further assessment of Human Error Analysis techniques and their value to the potential use of contracts to capture HF requirements is perhaps an area for future work.

3.0 PROPOSAL - HUMAN FACTORS CONTRACTS

3.1 Recap of What We Gain From the Literature Search

3.1.1 Section 3 records the literature search carried out in order to ascertain the history and current uses of contracts within the design and safety disciplines. Additionally, Section 3 aimed to establish which Human Factors Engineering (HFE) activities are employed early in the system design lifecycle and whether or not their outputs could be useful for my concept of HF contracts. In this section, I shall summarise the earlier findings, drawing out the safety contributions from the HF techniques studied with the aim of developing an integrated Design and Safety strategy using contracts to record the dependencies identified at each stage.

3.1.2 A contract is an enforceable agreement between two parties. The concept of contracts was first used in the software engineering domain by Bertrand Meyer. His ‘Design By Contract’ methodology encourages the use of assertions; pre- and post- conditions in the source code that establish the contract (*rely conditions*) between software components. These can be tested, but more importantly this method has the great advantage that it makes programmers think about the necessary constraints between components. Overall, Meyer’s approach aids in the production of reliable, robust and reusable software modules, but doesn’t explicitly add to current safety engineering processes.

3.1.3 In looking at current applications of contracts and contract-like strategies, it became apparent that contracts are being used to record dependencies at different levels of abstraction. Hierarchical contracts record the rely conditions between different levels of abstraction within a system model, and are also used to record the constraints between the applications and the OS and between the OS and the hardware on the Integrated Modular Avionics (IMA) architecture. Peer contracts record inter-dependencies between elements at the same level of abstraction, such as those between the separate IMA applications (e.g. between the Fuel Control and the Centre of Gravity modules).

3.1.4 The aforementioned IMA architecture offers discrete avionics applications running on a distributed computing platform. This offers the desirable property of long-term supportability via ease of upgrade, however this cannot be supported by current safety assurance processes as they produce monolithic safety cases that need expensive rework to accommodate even small changes. Bate, Hawkins and McDermid’s ‘Contract-Based Approach to Designing Safe Systems’ (2003 [10]) presents a strategy that identifies which modules contribute to the system hazards and hence allows constraints to be established – the contracts – to prevent the hazards occurring. This work was continued by Bate, Kelly et al (2003)[11] in order to develop a modular safety case strategy to compliment the IMA architecture.

3.1.5 With respect to the HF side of the literature survey, Work or Task Analysis encapsulates a variety of techniques aiming to analyse existing work/tasks to establish the user and organisational goals and the interactions between the involved parties. The use of subject matter experts, i.e. the users, to inform this process is invariably critical to the eventual usability and hence acceptance of the product. SME participation must also enhance the safety of the eventual product by eliciting dependencies between differing levels within an organization and across the user-system divide which might not otherwise have been revealed.

3.1.6 Vicente's Cognitive Work Analysis (1999) states five areas to be analysed in order to inform the design process optimally. The Work Domain contributes context constraints; Control Task analysis produces a solution-independent task description; Strategies provide further design guidance without actually prescribing the solution; Social and Organisational analysis captures higher-level interactions; and Worker Competencies must be established to inform the recruiting and training requirements and the safety case.

3.1.7 Function Allocation via techniques such as Hierarchical Task Analysis (HTA) and Use Cases produces a greater level of detail, due to their ability to model specific 'real life' scenarios as opposed to an abstract 'laboratory rat'. Function allocation is aided by the human and machine strengths and weaknesses contained in Fitts' List, and overall helps the designers understand the relationships between the system actors as well as shaping the design with respect to 'who does what'.

3.2 Evaluating the HF Analysis Data Set

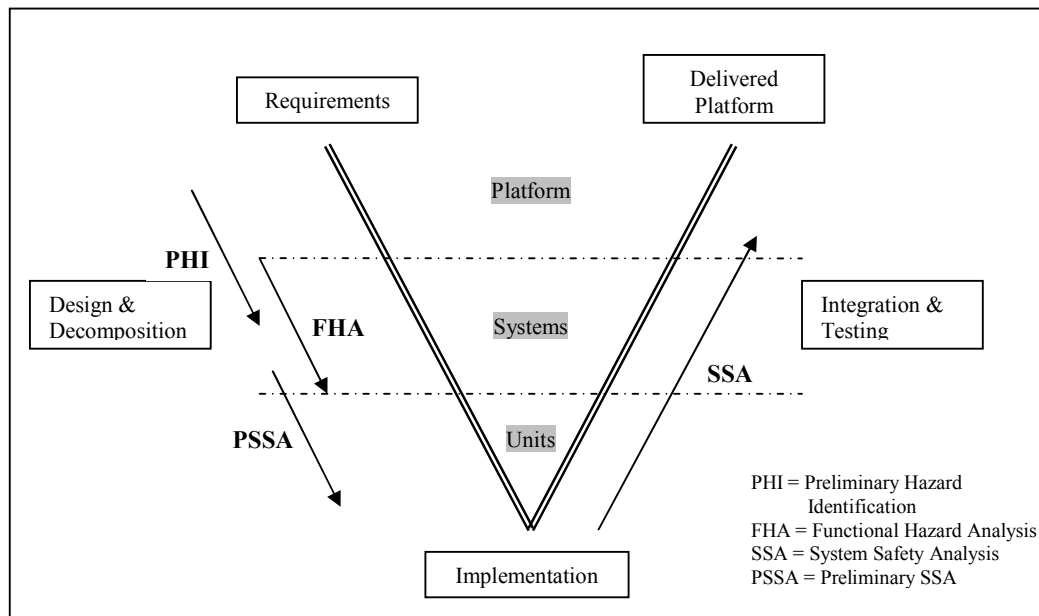
3.2.1 Continuing the recap theme, it is the aim of this paper to ascertain whether or not a contract-based approach can be used to capture the often-implicit dependencies between the designers of a system and its operators. The HF-related techniques described above have been shown to produce useful output, when applied early in the design lifecycle. The following paragraphs aim to evaluate this data set to see if it is not only useful, but essential. If this is the case, then contracts may be an appropriate strategy to capture the data set and make it available for further use later in the design and safety lifecycles.

Table 5 - Summary Table Showing Where Current HF Techniques Could Be Applied In The Design Lifecycle And What They Could Contribute To The Safety Process

<i>Process</i>	<i>Technique</i>	<i>Lifecycle Stage</i>	<i>Safety Product</i>
Domain Analysis	Checklists, User Interviews	Requirements	Fixed environmental constraints (e.g. operating temperature, altitude etc.) feeding the contextual model and hence the initial design, PHI/PHA and safety case.
Control Task & Strategy Analysis	Analysis of Statement of User Requirements, User and Stakeholder Interviews	Requirements	Broad system-level goals, feeding the initial design, PHI/PHA.
Social Organisation & Cooperation	Workplace observation, User and Stakeholder Interviews	Requirements	HF aspects of the contextual model (workplace inter-dependencies, current task allocation, procedure-following performance, attitudes to IT and automation, quality and safety etc.) Management and user-level goals. Feeds initial design and PHI/PHA processes.
Worker Competencies	Workplace observation, User and Stakeholder Interviews	Requirements	User capabilities and stakeholder expectations for the contextual model, informing future recruiting and training policies and the safety case.
Task Decomposition	Hierarchical Task Analysis	Requirements / Preliminary Design	More detailed user goals and achievement strategies established, allowing rudimentary function allocation and human error analysis.
Scenario Analysis / Function Allocation	Use Case Decomposition	Preliminary Design	Detailed function definition allowing function allocation and feeding the Functional Hazard Analysis processes. Also contributes scenario pre- and post-conditions, and possible exception scenarios (i.e. possibly hazardous failure conditions).
Human Error Analysis	Pessimistic task failure probabilistic estimates based on SME experience	Preliminary Design	Used to model human mitigation efficacy in accident scenario analysis through Event Tree Analysis for the production of pragmatic equipment safety targets.

3.2.2 Table 5 above contains a summary of the HF processes and techniques examined in Section 3; some of the data regarding the use of the output of the HF processes examined is based on the assumption that the system being designed is a direct replacement for an existing system. The table depicts my impression of the earliest juncture where each of the HF techniques might be applied within the generic combined design and safety process lifecycle model shown at Figure 8 below, along with the useful outputs from each technique.

Figure 8 - Integrated Design and Safety 'V' Lifecycle Model



3.2.3 In order to attempt to bring order to the seemingly chaotic nature of Table 5, I have categorised the HF process outputs as shown in Table 6 below. Upon evaluating the data presented in these tables, I conclude the following with respect to the suitability of the HF process output for contract formation:

- a. Firstly, the data *is* of value, either for informing the design and safety assessment processes further, for direct use in the safety case, or for informing the management processes required to operate the system. Hence the data should be recorded, which potentially could be achieved by some form of contract strategy.
- b. Further to the above, each member of the HF process output data set represents dependencies that *must* be maintained. The following examples illustrate this, hence reinforcing my intent to capture the dependencies via contract:
 - 1) The Domain Analysis produced a list of Non-Functional Requirements for the existing system. If the new system is being developed as a direct replacement, then these NFRs must be respected within the system design otherwise the machine elements and human operators may not be able to function in the workplace. Similarly, the procuring organisation, which I shall refer to as the Operating Authority (OA) henceforth, is bound to operate the new system in the same (or equivalent) environment, otherwise the developer, henceforth referred to as the Design Authority (DA), will not be responsible if

the system malfunctions. This therefore represents a two-way contract between the OA and the DA.

2) The Task Analysis produced use case scenarios indicating the roles that the human and machine actors play in the current system. This model informs the design process and hence the system safety analysis (SSA). Should the DA produce a system that doesn't reflect the current function allocation strategy, then the OA would have grounds to reject the product (again based on the assumption that the new product is a direct replacement for the current one). This again is a two-way dependency, since the DA could not be responsible if the OA decides to operate the system in a significantly different manner to its predecessor (e.g. with a watch of two ATC operators instead of five).

3) The Competency Analysis produced direct evidence for the safety case, for example supporting a claim that 'operator and maintenance personnel are sufficiently competent and experienced'. This represents commitments from the OA to the DA and to the public, since the safety case will be invalid and operations may be unsafe if the DA does not provide such competent personnel to operate the new system.

Table 6 - Table Showing Taxonomy for Early HF Process Output

<i>HF Process Output</i>	<i>Useful For Which Subsequent Process</i>	<i>Category</i>
Contextual Information	Design (Non-Functional Requirements). Safety Case.	Design-Feeding (DF). Safety Case Evidence (SCE).
Task Descriptions	Design (Task / Functional Requirements). Functional Hazard Assessment (FHA). Probabilistic Human Error Analysis (HEA).	DF. Safety Analysis-Feeding (SAF). SAF.
Personnel-Focused HF Contextual Information (e.g. workplace interdependencies, current task allocation, attitude to IT and automation, QA and Safety Management).	Job design for users of the new system (see Note 1 overleaf).	Ops Management (OM).
User Competencies.	Operating Requirements (Personnel and Training Strategies – see Note 2 overleaf). Design (Function Allocation). Safety Case.	OM. DF. SCE.

Note 1 – Although not directly supporting the safety process, the detailed involvement of the existing system users in the job design for the users of the new system should bring about significant improvements in the work-related Performance Shaping Factors / Error Producing Conditions (including job satisfaction and management approval rating). This in turn will benefit the business as a whole, including safety performance.

Note 2 – Care must be taken not to blindly base personnel competency and training requirements from analysis of the users of the current system. Even if the new system is a direct ‘fit, form and function’ replacement for the current one, it is likely to need new skills to operate and maintain it, especially if the current system uses significantly older technology.

3.3 HF Contract

3.3.1 In the previous section, the data set of the outputs from the HF processes under study were categorised and assessed. This process concluded that the data set represented essential HF-related dependencies that should be captured in order to support further the design and safety assurance activities for the new system. Due to the nature of the dependencies described above, and to parallels that can be drawn with the uses of contracts described earlier in the paper (see Section 2.2 above), I propose to use a simple tabular format to represent these HF Contracts.

3.3.2 Table 7 below shows a fictitious example of a HF contract. In this instance, it is being used to capture a supervisory requirement for a military Air Traffic Management system. The system Operational Safety Analysis (OSA) included Task Analysis within the Operations Room, which was used to inform an Event Tree Analysis of the hazard ‘mid-air collision’, leading to the requirement that Fighter Allocators (senior controllers) must not be responsible for more than two Weapons Controllers (junior controllers). This source analysis is referenced from the HF contract, as are any related hazard log entries and system requirements. The contract creation date, its name and a unique reference number should also be allocated. Clearly this template would lend itself to integration with hazard and requirements logging tools such as CASSANDRA[23] and DOORS[24], however this is beyond the scope of this paper and is not discussed further.

Table 7 - Example (Fictitious) HF Contract

<i>Contract Number:</i> OM001/05	<i>Supplier Name:</i> MOD
<i>Date of Contract Formation:</i> 1 Sep 2005	<i>Client Name:</i> IBM Farnborough
<i>Contract Name:</i> UCCS Supervisory Requirement at FA Level.	<i>Contract Type:</i> Operations Management (OM)
<i>Contract Details:</i> The MOD shall ensure that a Fighter Allocator (FA) supervises no more than two Weapons Controllers (WCs).	
<i>Source Activity:</i> Operational Safety Assessment (OSA)	

<i>Link to Requirement Number:</i>	N/A
<i>Link to Hazard Number:</i>	OH12 Controller Workload

3.4 HF Contract Maintenance

3.4.1 It can be foreseen that, even for a small safety-related system, an extensive early application of HF analyses, as proposed in this paper, might produce a large number of HF contracts. Conmy, Nicholson and McDermid (2003)[12] and Bates, Bate et al (2003)[11] both raised this issue without proposing solutions. There is no simple solution to this, however due to the very nature of safety contracts, the issues of configuration control and ‘door-stop avoidance⁴’ need to be addressed.

3.4.2 To enable the contracts to be sorted and thereby managed in smaller chunks, I have included a space to record the *contract type* on the template. I suggest using my scheme from Table 6 above (DF, SCE, SAF and OM), however other taxonomies might prove more suitable for different projects. Tool support would also ease the administrative burden and facilitate distributed access, which would be advantageous considering the diverse range of disciplines that could need access to the data set, especially in the early design phases.

⁴ How to keep a large volume of technical documentation in its intended use, instead of its more likely occupation of keeping the office door open on a hot day.

4.0 CASE STUDY

4.1 Introduction.

4.1.1 In order to test my strategy for identifying implicit HF dependencies and capturing them using HF contracts using a real safety-related design and development undertaking, I looked to my varied portfolio of military equipment procurement projects. Regretfully, I could not get access to anything suitable at the right lifecycle stage, i.e. early enough in the requirements or preliminary design processes, that had any significant HF work either completed recently or ongoing.

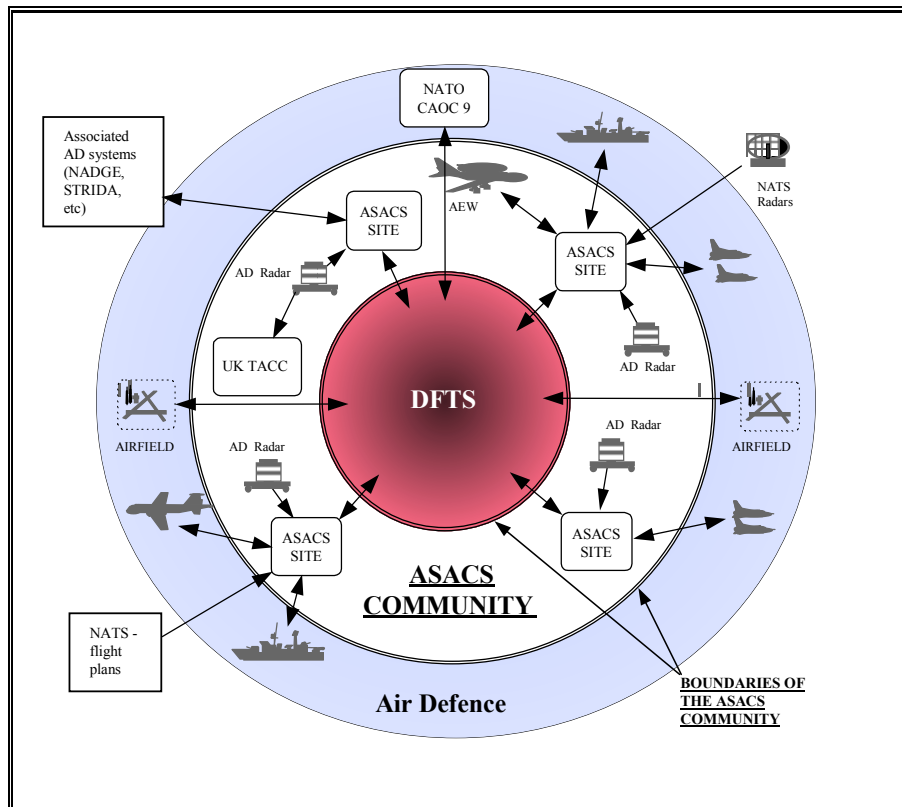
4.1.2 Not to be deterred, I have carried out a retrospective analysis of a mature MoD system, the UK Air Surveillance and Control System (ASACS), in order to see whether a set of HF dependencies can be extracted that would be equally as suitable for capture using HF contracts as was the theoretical HF data (sub-) set⁵ obtained in Section 3.0 above. The following paragraphs provide an in-depth introduction to the ASACS in order to give an impression of the complexity of the technical and human scope of the system.

4.1.3 The Air Surveillance And Control System (ASACS) provides the UK military (predominantly the RAF - Royal Air Force) with Control and Reporting facilities for use within the UK Air Defence Region (UKADR – airspace above and extending over the seas around the UK, as defined by NATO) and at overseas locations when the military is deployed on peacekeeping or air policing operations. *Control* refers to the control of weapon systems (principally fighter aircraft and short-range air defence assets (SHORAD)) and supporting aircraft (e.g. AAR and AEW - air-to-air refuelling aircraft, or ‘tankers’, and airborne early warning). *Reporting* refers to all of the activities involved between detecting the presence of an aircraft and classifying it as a valid air track, namely *surveillance, detection, identification and reporting*.

4.1.4 The ASACS has its beginnings in the Second World War Air Defence (AD) network, comprising the Chain Home coastal radars and centralized Control and Reporting Centres (CRCs). This has been developed and modernized successively following the same concepts through the Sixties-vintage UK Air Defence Ground Environment (UKADGE) and onto today’s primarily digital ASACS, described further in the following paragraph.

⁵ I have used the term sub-set here since the HF analysis techniques considered earlier are not an exhaustive compilation.

Figure 9 - ASACS Conceptual Diagram



4.1.5 Figure 9 above, is a conceptual diagram of the ASACS showing some of the elements contributing to this complex system. The inner core is the Defence Fixed Telecommunications System (DFTS), which is a leased secure digital telecommunications network used as the bearer for all terrestrial ASACS data. The outer layer depicts the military weapons and support elements with which the ASACS must interact. Beyond this are the non-military or allied military organizations and systems that interface with the ASACS; these include the receipt of radar data from some of National Air Traffic Services' (NATS) air traffic control (ATC) radars, and the interchange of aircraft plot data from the NATO and French AD networks. The ASACS band itself comprises the following:

- a. UCCS - The UCMP Command and Control System. The UCCS is a robust, networked command and control (C2 or C²) system, the usual architecture being two Control and Reporting Centres (CRCs) in a main and standby configuration. The other main network elements are the two former Control and Reporting Posts (CRPs), which retain an UCCS CCS but no controller positions, and the NATO Combined Air Operations Centre (CAOC), which also has a CCS and from which NATO operations executives detached from Headquarters AIRNORTH in Germany manage the network configuration. The CCSs route the data between the major system elements, using the DFTS as the bearer as mentioned previously. The CCSs at the CAOC and each CRC compile aircraft tracks into the Surface and Recognized Air Picture (SRAP) from the data being received from the network. The CCS also stores locally entered "tote" data (meteorological data, squadron strengths etc.) as part of UCCS' Resource Data

Catalogue (RDC). Finally, UCCS provides the fighter controllers with situational data (user-filtered aircraft plots and tracks on a large graphical TFT display), support data (RDC on two smaller TFTs), and communications services (ground-to-air and ground-to-ground voice) via the Voice Communications System (VCS). For selection and data input, fighter controllers are provided with a rolling ball pointing device, a traditional QWERTY keyboard, and a modal special functions keyboard.

b. Air Defence Radars. The AD radars are the primary source of aircraft plot data for the UCCS, each providing, via landline (although the capability exists to use point-to-point microwave links), a digital data stream containing both primary, secondary and combined plots (processed returns from radio-frequency energy reflected back from aircraft to the radar, active returns from aircrafts' secondary surveillance radar (SSR) transponder, and correlated primary and secondary plots respectively). Four different types of military radar currently feed into the UCCS via their nearest CCSs. One type is heavily automated and can be adjusted remotely, however the processing parameters on the others need to be regularly updated by local operations staff, known as Tactical Radars Operators (TACROs) in order to compensate for local weather and propagation conditions and thereby produce optimized data for the controllers in the CRCs. The semi-permanently sited radars are designated the Type 91 (T91), T92 and T93, however ASACS' other military radar type, the T101, is a fully mobile, tactical, and air transportable system that can be deployed away from the UCCS to operate autonomously or in support of a deployable Tactical Air Control Centre (TACC).

c. E3D Airborne Early Warning Aircraft. The E-3D AEW aircraft operated by the RAF are self-contained platforms capable of operating autonomously, or of linking into the UCCS or a dynamic air/surface network via secure digital data links. The E-3D has its own SSR and primary AD radars mounted above the fuselage giving 360° of lateral coverage, and good vertical coverage down to ground level. In general, the E-3D controllers are able to offer the same level of aircraft control as their ground-based ASACS colleagues, however some restrictions are applied below 10,000ft (see ATS at paragraph 4.1.6 below).

d. UK Tactical Air Command Centre (UK TACC). The RAF also operates an air-deployable TACC. This is used in conjunction with one or more of the T101 radars to provide a robust ATS capability in support of UK and allied commitments overseas. UK-based TACC ATS operations also fall within the scope of the ASACS.

4.1.6 Air Traffic Services (ATS) and Airspace Issues. To understand the different levels of aircraft control exercised by the ASACS, and thereby the differing levels of responsibility associated with each one, it is first necessary to clarify the different categorizations of airspace within which the ASACS must operate. Due to the number of passengers carried, collision with, or unintended engagement of, *commercial air transport* (CAT) represents the largest risk to the ASACS. In order to provide increased protection for CAT against mid-air collision, an internationally agreed airspace categorization scheme for *controlled airspace* (CAS) has been established. Minimum equipment standards and crew competencies are established for aircraft using CAS, and ATC providers are mandated to apply radar services which require that certain categories of aircraft are physically separated by vertical and horizontal distances which meet or exceed laid down minimum separation criteria. Within the UK Flight Information Regions, CAS includes corridors designated as airways, control zones, and control areas at the

confluence of airways and surrounding major civil airports, together with most of the airspace above 24,500ft (FL245). For aircraft operating in *uncontrolled airspace*, i.e. outside of CAS, it is the pilots' responsibility for collision avoidance using the basic premise of 'see and be seen'. For civil aircraft outside CAS, there are maximum speed restrictions of 250kts below 10,000ft however this does not apply to military aircraft, which may operate in uncontrolled airspace sub-sonically over land and supersonically over the sea.

4.1.7 The radar control services executed by ASACS fighter controllers are as follows:

- a. Radar Control (RC). Radar Control is the mandatory service implemented by ASACS controllers for aircraft operating in, or transiting through, CAS. When providing this service, the ASACS controller bears responsibility for ensuring that the prescribed minimum aircraft separation distances are met or exceeded by issuing mandatory instructions to the military aircraft under control.
- b. Radar Advisory Service (RAS). Due to limitations in radar coverage at lower levels and over mountainous regions, it is not possible to provide RC outside CAS. However, at the pilots' request, ASACS controllers can provide a Radar Advisory Service, whereby, according to the ASACS Safety Case [21] "the controller is obliged to provide advisory vectors and/or level instructions necessary to achieve separation from other traffic of at least 5nm or 3000ft." The pilots are not legally obliged to follow the instructions given by the ASACS controllers under RAS, however if they chose not to do so then they are responsible for any avoidance action that may be required.
- c. Radar Information Service (RIS). A lesser service that may be provided by ASACS controllers to pilots operating outside CAS is the Radar Information Service (RIS). When providing a RIS, the controller is obligated to provide comprehensive, accurate, and timely information to pilots of any aircraft that might potentially present a risk of collision. This is in order that pilots can focus their own visual detection in a certain direction so that they can take their own avoidance action.
- d. Flight Information Service (FIS). A Flight Information Service is not a radar control service, but merely information that might prove helpful to military pilots carrying out operations or training. Examples of such information are meteorological updates, airfield condition states, or any other information that might have safety implications. For a FIS, the controller is not required to continuously monitor the flight and is not responsible for maintaining the safe separation of aircraft. Circumstances when the ASACS might be required to offer this reduced service might be, for example, due to technical limitations, i.e. because of equipment unavailability or when new hardware or software builds are being introduced.

4.1.8 MoD and Contractor 'Actors'. As described at section 4.1.3 above, the ASACS' role has always been, and remains today, the command and control of defensive aircraft and operations. Because of the nature of the weapons systems employed, namely fighter aircraft and surface-to-air missile (SAM) systems, the whole of the ASACS has been historically lodged within the RAF's 'Strike Command' (STC). All of the RAF's operational responsibilities currently lie within STC, with differing functional responsibilities organized administratively in the smaller 'Groups'. The AD fighter aircraft are functionally located

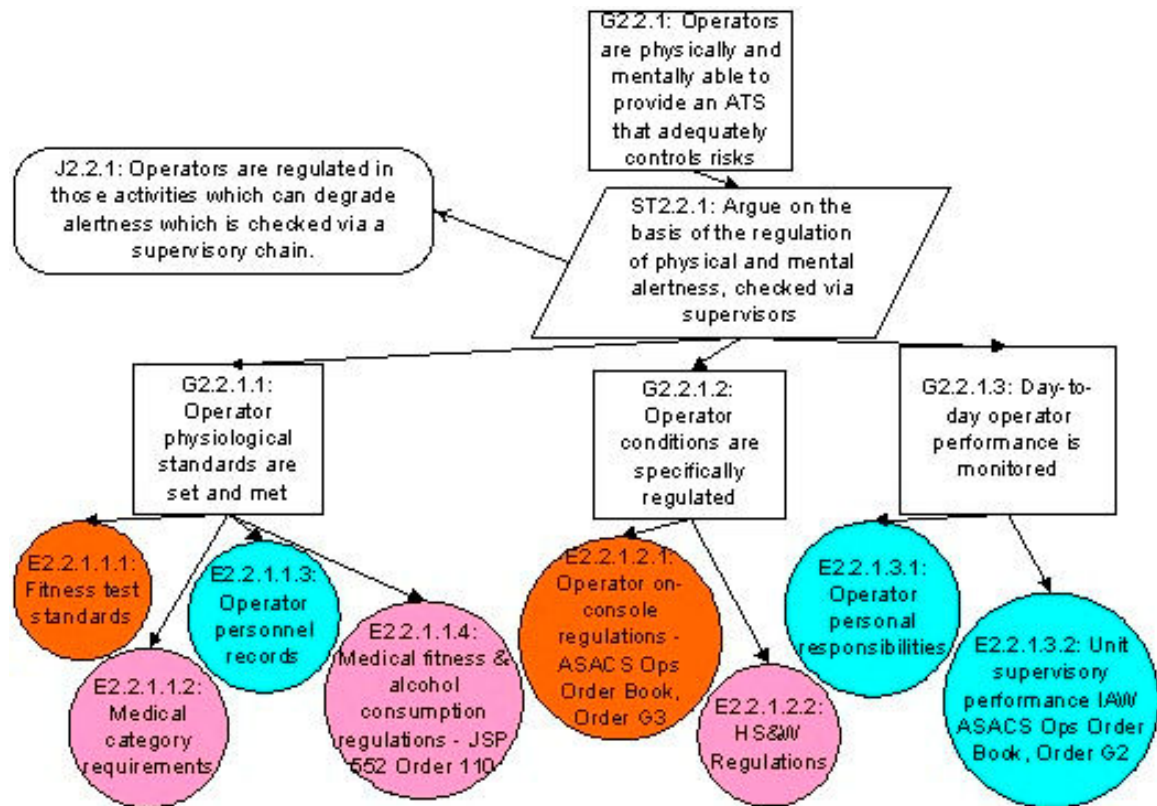
within Number 1 Group, and the ASACS ground elements and E3D AWACS aircraft are within Number 3 Group. The RAF provides all of the fighter controllers that operate using the UCCS and the TACC, as well as the technicians that carry out most of the ASACS preventive and corrective maintenance. Budget constraints are driving ever more demanding efficiency measures, and this has led to an increasing number of maintenance activities being carried out under *Contractor Logistics Support (CLS)* arrangements. IBM is the DA for the UCCS, and they hold the CLS contract covering a significant portion of the UCCS corrective maintenance activities.

4.2 ASACS Safety Case

4.2.1 As the ASACS comprises such a disparate mix of systems (it is often referred to as a ‘supra-system’), an overarching Operational Safety Assessment known as the ASACS Safety Case (Issue 1)[22] was put in place in 2001 to cover all UK ASACS operations. This has just been up-issued to Issue 2 [21] to take into account the many system and organizational changes that have taken place in the interim period.

4.2.2 Figure 10 below is a small portion of the argument structure from the ASACS Safety Case Issue 2. It depicts one of the HF-related arguments and is included to give an impression of the depth of analysis contained within the document.

Figure 10 - Excerpt From The ASACS Safety Case



4.3 ASACS HF Contracts

4.3.1 I introduced my intent to derive HF contracts from the RAF's ASACS at the beginning of Section 4.0 above. The approach that I took for this was a mix of interviewing the main ASACS Safety Case author (who wished to remain anonymous), and a detailed analysis of the document itself. This resulted in Table 8, which uses the categorisation scheme developed previously, but slightly modified to allow discrimination between requirements to be satisfied at an ASACS-site level ('local') and those to be satisfied at the higher 'System' level.

Table 8 - ASACS Human Factors Related Dependencies

<i>Category</i>	<i>Level</i>	<i>Requirement</i>	<i>Rqt Source</i>	<i>Source Analysis</i>
OM	System	System-wide Configuration Control	Users	OSA
OM	Local	Local Software / Adaptation Configuration Control	DA	SSA
OM	System	Training: the provision of sufficient trained Ops and Eng personnel (recruiting and training policies)	DA	SSA
OM	Local	Training: ensure that Ops and Eng personnel retain Currency	DA	SSA
OM	System	Management of Contracted-out Services (maintenance, provision of supporting data e.g. maps and meteorological data).	User	OSA
OM	System	Provision of System Operating and Maintenance Procedures	DA	SSA
OM	System	Provision of Operations Procedures	User	OSA
OM	Local	Provision of Motivated Personnel	User	OSA
OM	System	Provision of Alert, Fit and Able Personnel (shift patterns, staff levels, fitness policy)	User	OSA
OM	Local	Provision of adequate Supervision for Ops and Eng activities	User	OSA
OM	System	Safety Management System employed to monitor and improve safety performance	User	OSA
OM	Local	All Maintenance carried out in accordance with Procedures (supervision / QA system)	DA	SSA
OM	System	Contractor Logistics Support meets contracted requirements	User	OSA
OM	System	Corrective Maintenance carried out within required timescale	DA	SSA
OM	Local	Maintenance carried out using recommended spares and materials	DA	SSA
OM	Local	Environment controlled and kept suitable for operators and equipment (heating, lighting, ventilation, noise and cooling).	DA	SSA
OM	Local	Safety-related information available and up to date (map overlays, communication plans, scheduling information etc.)	User	OSA

4.3.2 From the table above, it can be seen that a retrospective ‘paper-based’ analysis such as this is still able to capture HF-related dependencies. In my initial categorisation of the data set, I extended the taxonomy to include *Maintenance* and *Training* requirements and used these to describe a number of the table entries. However, upon subsequent review of the table, I re-evaluated my decision and made several changes that left the entire set comprising only Operations Management (OM) category dependencies. This is likely to have occurred for the following reasons:

- a. Firstly, the retrospective nature of the analysis, in that the ASACS is a mature system, and my analysis was based upon people populating and documentation predominantly concerning the user community.
- b. Secondly, that the ASACS Safety Case Issue 2 has purposely avoided extensive analysis of equipment failures in favour of the elements that are most likely to cause incidents, namely the human actors: the operators, technicians and both local and system-level management.

4.3.3 Further to the above, it is clear that the HF-related dependencies in the table could be expressed using the proposed HF contract template. To fill several pages with examples of these would be nugatory effort, however, it would be interesting to elicit the opinions of the ASACS local Unit Safety Management Officers as to whether a set of such HF contracts would be more usable for day-to-day safety management purposes (occurrence investigation, targeted safety management surveys etc.) than the ASACS Safety Case.

5.0 FURTHER WORK

5.1.1 The literature survey commencing at Section 2.5, researching HF techniques to be included in this methodology, was limited in scope due to time constraints. Possible expansion of this paper could include consideration of a wider range of techniques, and the provision of a systematic approach advising which techniques to use in different situations, for example for systems with more or less stringent certification requirements.

5.1.2 One of the observations in Section 3.0 mirrored that made by other researchers into the uses of contracts in safety-related activities, namely how to manage the volume of contracts that are created. Making the contracts accessible to all of the parties involved in the project is vital, and this suggests a network-based contracts management tool. However, other aspects such as configuration control activities also need to be addressed.

5.1.3 Following on from the previous administrative point, I have not had the resources to examine the implications of potential dependencies within the set of HF contracts. It is likely that such dependencies will exist, for example any contracts drawn out from Workplace Analysis will be linked by the actors in the workplace, and hence a more detailed study into this aspect might be worthwhile and might add value to more than just the proposal in this paper.

6.0 CONCLUSIONS

6.1.1 The aim of this project was to develop a strategy to help integrate HF techniques into the established system design and safety engineering processes. It has been shown that the early employment in the system development cycle of a range of HF analysis techniques can generate essential data, and that this data should be captured to feed and support the design and safety processes. Furthermore, it has been shown that a simple tabular HF contract format can be used to record the set of HF-related dependencies that are the product of the proposed strategy. Finally, further work that might be undertaken in future to broaden this process has been suggested.

7.0 REFERENCES

1. Neil Storey “Safety-Critical Computer Systems”, Addison-Wesley 1996.
2. Online dictionary: www.Dictionary.com
3. Fran Przyblewski article “Contract Elements – Contract Formation” available online on the ‘SeniorMag’ website at: <http://www.seniormag.com/legal/contractelements.htm>
4. International Electrotechnical Commission: “IEC 61508 – Functional Safety of Electrical/Electronic/Electrically Programmable (E/E/PE) Safety-Related Systems”, available online at <http://www.iec.ch>
5. Bertrand Meyer’s ‘Applying “Design By Contract”’, IEEE Computer, October 1992 (available online at: <http://www.cs.umb.edu/~jxs/courses/2004/681/papers/meyer-computer.pdf>).
6. Jean-Marc Jezequel and Bertrand Meyer’s article “Design by Contract: The Lessons of Ariane”, IEEE Computer, January 1997 (also available online via the ISE website at: <http://archive.eiffel.com/doc/manuals/technology/contract/ariane/page.html>).
7. Bill Venners and Bertrand Meyer: “Design by Contract – A Conversation With Bertrand Meyer, Part II”, 8 December 2003 (available online at: <http://www.artima.com/intv/contractsP.html>)
8. Antoine Beugnard, Jean-Marc Jezequel, Noel Plouzeau and Damien Watkins: “Making Components Contract Aware”, IEEE Computer, July 1999 (available online at: <http://doi.ieeecomputersociety.org/10.1109/2.774917>)
9. P.M. Conmy, M. Nicholson, Y. Purwantoro and J.A. McDermid: “Safety Analysis and Certification of Open Distributed Systems”, available online at: <http://www-users.cs.york.ac.uk/~mark/papers/UCCS02.pdf>
10. I.J. Bate, R.D. Hawkins and J.A. McDermid: “A Contract-Based Approach to Designing Safe Systems”, Australian Computer Society Inc, appearing at the 8th Australian Workshop on Safety Critical Systems and Software (SCS’03), Canberra. Conferences in Research and Practice in Information Technology, Vol 33. Also available online at <http://www-users.cs.york.ac.uk/~rhawkins/papers/OzPaper.pdf>
11. S.A. Bates, I.J. Bate, R.D. Hawkins, T.P. Kelly and J.A. McDermid: “Safety Case Architectures to Complement a Contract-Based Approach to Designing Safe Systems”, presented to the 21st International System Safety Conference 2003, and available online at <http://www-users.cs.york.ac.uk/~rhawkins/papers/SimonISSC.pdf>
12. P.M. Conmy, M. Nicholson and J.A. McDermid: “Safety Assurance Contracts for Modular Architectures”, Australian Computer Society Inc, appearing at the 8th Australian Workshop on Safety Critical Systems and Software (SCS’03), Canberra. Conferences in Research and Practice in Information Technology, Vol 33. Available

- online at <http://crpit.com/confpapers/CRPITV33Conmy.pdf>
13. Adelard “ASCAD – Adelard Safety Case Development Manual”, available online at <http://www.adelard.co.uk/resources/ascad/>
 14. Object Management Group (OMG) “Unified Modelling Language” – see online via <http://www.uml.org> and <http://www.omg.org>
 15. Ministry of Defence “Interim Defence Standard 00-56, Safety Management Requirements for Defence Systems, Part 1: Requirements” Issue 3, 17 December 2004, available from UK Defence Standardization: <http://www.dstan.mod.uk>
 16. K.J. Vicente “Cognitive Work Analysis: Towards Safe, Productive and Healthy Computer-Based Work”, 1999, Lawrence Erlbaum Associates.
 17. P.M. Fitts “Human Engineering for an Effective Air Navigation and Air Traffic Control System”, 1951, Washinton DC: National research Council.
 18. Alistair Cockburn “Writing Effective Use Cases”, 2001, Addison-Wesley.
 19. R. Barraclough “‘OUSE.COM’ SOFTWARE FUNCTIONAL REQUIREMENTS”, Oct 2002, prepared for University of York Safety Critical Systems Engineering MSc Requirements Engineering (RQE) Open Assessment.
 20. N.G. Leveson “Safeware, System Safety and Computers, A Guide to Preventing Accidents and Losses Caused by Technology”, 1995, Addison-Wesley Publishing Company Inc.
 21. Royal Air Force - HQ 3 Gp Airspace Control Capability Safety Management “HQ 3 Gp Air Surveillance and Control System (ASACS) Safety Case – Issue 2.0”, Aug 2005.
 22. Royal Air Force - ASACS Safety and Standards Unit “HQ 2 Gp Air Surveillance and Control System (ASACS) Safety Case (2GP/1480/8/ASACS) – Issue 1.0”, 21 June 2001.
 23. HVR Consulting “CASSANDRA Hazard Management Tool”, available from <http://www.risktools.co.uk/cassandra.htm>
 24. Telelogic “Dynamic Object-Oriented Requirements System (DOORS)”, available from <http://www.telelogic.com>