

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

**A DECISION SUPPORT SYSTEM FOR NAVAL
AVIATION MISHAP INVESTIGATION AND
REPORTING**

by

Charles E. Emde

September 1995

Thesis Advisor:

Hemant K. Bhargava

Approved for public release; distribution is unlimited.

19960206 131

DTIC QUALITY INSPECTED 1

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE A DECISION SUPPORT SYSTEM FOR NAVAL AVIATION MISHAP INVESTIGATION AND REPORTING			5. FUNDING NUMBERS	
6. AUTHOR(S) Emde, Charles E.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) <i>This thesis discusses the implementation of a prototype of a decision support system (DSS) for aviation mishap reporting. The Naval Aviation Safety Program, as defined by OPNAVINST 3750.6Q is a complete reference for the reporting requirements for naval aviation mishap reporting, and this thesis augments the reporting requirements by defining a mishap investigation heuristic in the form of a logical model. This model directly addresses problems an investigator may encounter in the course of a mishap investigation such as logical omissions and incomplete deduction or investigation. Typically, mishap investigators are faced with numerous, unorganized pieces of evidence which develop into a complex web of interrelationships which recreate the events which caused the mishap. Our model suggests a process which organizes evidence and cause factors, and then we automate the model in a decision support prototype. The system also addresses the "administrative overhead" of a mishap by outlining the architecture of a complete system providing facilities for initial and final mishap reporting in addition to the automation of the deliberation model.</i>				
14. SUBJECT TERMS Decision Support Systems, Aviation Safety Reporting, Causal Reasoning, Semantic Reasoning			15. NUMBER OF PAGES 126	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Approved for public release; distribution is unlimited.

**A DECISION SUPPORT SYSTEM FOR NAVAL AVIATION MISHAP
INVESTIGATION AND REPORTING**

Charles Emde
Lieutenant, United States Navy
B.A., University of Colorado, 1985

Submitted in partial fulfillment
of the requirements for the degree of

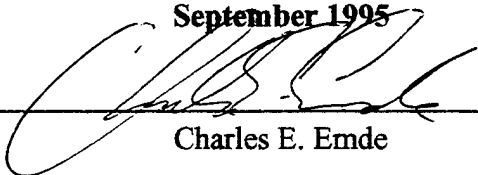
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL


September 1995

Author:

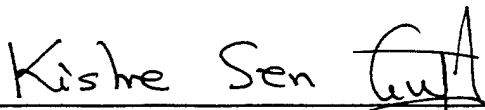


Charles E. Emde

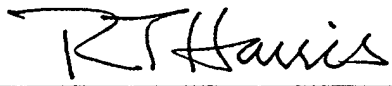
Approved by:



Hemant K. Bhargava, Thesis Advisor



Kishore Sengupta, Associate Thesis Advisor



Reuben T. Harris, Chairman
Department of Systems Management

ABSTRACT

This thesis discusses the implementation of a prototype of a decision support system (DSS) for aviation mishap reporting. The Naval Aviation Safety Program, as defined by OPNAVINST 3750.6Q is a complete reference for the reporting requirements for naval aviation mishap reporting, and this thesis augments the reporting requirements by defining a mishap investigation heuristic in the form of a logical model. This model directly addresses problems an investigator may encounter in the course of a mishap investigation such as logical omissions and incomplete deduction or investigation. Typically, mishap investigators are faced with numerous, unorganized pieces of evidence which develop into a complex web of interrelationships which recreate the events which caused the mishap. Our model suggests a process which organizes evidence and cause factors, and then we automate the model in a decision support prototype. The system also addresses the "administrative overhead" of a mishap by outlining the architecture of a complete system providing facilities for initial and final mishap reporting in addition to the automation of the deliberation model.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. PURPOSE	1
B. SCOPE	1
C. MISHAP CHARACTERISTICS	2
1. Uncertainty	3
2. Time Constraints	4
3. Administrative Overhead	4
D. SOLUTION	5
II. DETAILED DESCRIPTION OF THE PROBLEM	7
A. THE NAVAL AVIATION SAFETY PROGRAM	7
B. PROGRAM PARTICIPANTS	8
1. The Senior Member	8
2. The Aviation Safety Officer	9
3. The Aviation Mishap Board	10
4. Other Experts	11
C. MISHAP INVESTIGATION	11
1. The Mishap Investigation Report	12
D. THE INVESTIGATION PROBLEM	14
1. Example Presentation	14
2. Why is there the Need for an Automated Deliberation Process?	15
III. A DECISION SUPPORT SYSTEM FOR NAVAL MISHAP INVESTIGATION	19
A. SYSTEM OVERVIEW	19
1. General Description and Requirements	19
B. MODULE OR APPLICATION DEVELOPMENT	21
1. Server and Database Design	21
2. Initial Reporting	22
3. The Final Mishap Reporting Application	27
IV. THE DELIBERATION APPLICATION	29
A. BACKGROUND	29
1. The Aviation Safety School Model	29
B. THE AMB DELIBERATION MODEL	31
1. Model Description	33

2. The Deliberation Engine	40
3. Interface Goals, User Run-Time Routines	44
4. Screen Prototype Presentation	45
V. CONCLUSION	57
A. IMPLEMENTATION ISSUES	57
B. QUESTIONS FOR FURTHER STUDY	58
LIST OF REFERENCES	61
APPENDIX A	63
APPENDIX B	69
APPENDIX C	99
APPENDIX D	109
INITIAL DISTRIBUTION LIST	117

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to develop formal methodologies and software to improve the naval aviation mishap investigation and reporting process as described by the OPNAV 3710 series instruction. The use of information technology and a customized application will, it is hoped, improve the quality, timeliness and accuracy of an investigation by providing logical direction and reducing administrative overhead. The focus of our research is on supporting the deliberation process occurring during the mishap investigation process. This thesis will outline the requirements of such an application and provide an example of a basic prototype of the key element of the system.

B. SCOPE

The scope of this thesis encompasses the development of a Decision Support System that supports the Naval Aviation Mishap reporting process, concentrating on the development and implementation of a "deliberation model". Object oriented and Rapid Application Development technologies bring this type of development out of the professional software programming environment and put it in the hands of "power- users" and non- programmers. It is our intention to support the end- user computing environment: We describe an architecture which developers can easily implement and focus on the "deliberation" model and application which is unique research. Rather than focusing on the implementation of a prototype only, we also present the development of the underlying model. The implementation of the entire system is a topic in this thesis, but it's importance is secondary. Our primary goal is the development, formalization, and implementation of the mishap investigation deliberation process, and we present a basic implementation within an architecture for a complete client/ server system.

C. MISHAP CHARACTERISTICS

Naval aviation mishaps are the inevitable result of the practice of aviation. The Navy and other services continue to strive for a reduced aviation mishap rate through numerous safety programs. The Naval Aviation Safety Program is one such program responsible for reductions in aircraft accidents through the careful investigation, documentation and analysis of aircraft accidents. Mishaps occur for numerous reasons, and those reasons are a mystery until the evidence is collected and interpreted by a board of investigators.

The Naval Aviation Safety Program, OPNAV instruction 3750.6Q directs the conduct of naval aviation mishap investigations including the initial reporting of the mishap, the collection of evidence, and the formal publication of the final investigation and accompanying supporting evidence. The success of the Naval Aviation Safety Program is attributable to the Naval Safety Center's ability to organize and examine aggregate data collected from mishap investigations. Reduced mishap rates are the result of direct actions taken in response to mishap investigations, and to achieve this the investigations are highly structured and programmed. This high level of structure allows analysts to identify dangerous trends and mandate immediate changes to operational procedures and training when necessary. The mishap rate of 50 mishaps per 100,000 flight hours in the first reporting of accidents in the 1950s of has dropped to 1.96 in May of 1995 because of this ongoing analysis and data collection [Naval Safety Center Statistics].

Military aviation mishaps share the common elements of accidents and mishaps that occur in other contexts, but the military environment and the OPNAVINST 3750.6Q mandates place extraordinary pressures on the mishap investigator and investigative board. The most significant of these pressures explained in more detail below are uncertainty, time constraints, and the unique administrative burdens imposed on an Aviation Mishap Board (AMB) during the course of a mishap investigation.

1. Uncertainty

Uncertainty exists at many different levels in the mishap investigation. Obviously, the primary source of uncertainty in mishap investigation is the evidence presented with unanswered causes. It is the goal of the Aviation Mishap Board to deliberate upon the evidence and accurately identify the hazards to naval aviation defined by the evidence. Although the members of the Aviation Mishap Board are experts in areas such as aircraft operations and maintenance, they may lack the expert knowledge required to interpret, organize and investigate evidence presented by outside sources. The AMB is thus responsible for its own training as well as the tasks associated with active mishap investigation. The level of this training introduces additional uncertainty defined by the competence of the board. A required member of any Navy AMB is the Aviation Safety Officer (ASO) who is specially qualified to reduce this type of uncertainty by training the board members and "translating" expert data to facilitate deliberations.

An additional source of uncertainty is introduced by the nature of the mishap. Evidence left in the wake of a mishap may not immediately point to obvious causes but their careful organization may imply numerous event "chains" which the AMB must consider. Often those mishaps which leave little physical evidence become difficult to investigate because the number of causal factors cannot be eliminated based on evidence. In these cases an AMB must develop many different scenarios which point to sources of evidence and then deliberate on the validity of each. How these facts relate to each other and how the scenarios are inter-related produces uncertainties which require the AMB to have an organized strategy for organizing and recording deliberations. An AMB which proceeds with an investigation without a strategy for documenting its own deliberations will revisit uncertainties and questions numerous times resulting in inefficiencies and inaccuracies in logic. The central topic of this thesis, the deliberation model, directly addresses the above discussed uncertainties. A supporting application development is suggested around this model to address other important problems faced by the AMB.

These include time constraints and administrative overhead. Other problems more specific to the deliberation model are discussed in Chapter II of this text.

2. Time Constraints

The timely delivery of the results of an investigation are critical to the prevention of similar mishaps. As such, OPNAVINST 3750.6Q requires completion of investigations of specified severity within given time periods. Time constraints force additional pressures on the deliberation process, and poorly managed mishap boards may produce inaccurate investigations. Although the Mishap Investigation Report is a highly structured document which forces the authors to make logical connections between evidence and causal factors, the Naval Aviation Safety Program instruction does not suggest a clear strategy for board deliberation. Without such a strategy, the time constraints placed on the investigating body not only tax the organizational structure of the board and the personal skills of the ASO, but they may also affect the quality of the investigation. The deliberation model developed in this thesis, together with the suggested additional modules of the decision support system, will aid the AMB in reaching timely, logically accurate conclusions. Although not implemented here, the mishap reporting process can be greatly enhanced by the implementation of the database portion of the decision support system. This implementation would automate many of the message reporting procedures making the time constraints placed on a mishap board less significant.

3. Administrative Overhead

The Secretary of the Navy assigns the mishap reporting and investigative tasks to the "reporting custodian" of the aircraft involved in the mishap [OPNAVINST 3750.6Q].

Safety instructions and directives require the existence of a standing mishap board, but often squadron or activity resources do not allow the assignment of the administrative support necessary for a major investigation on a permanent basis. As a result, the administrative burden associated with the mishap investigation process can further contribute to the quality of the investigative results as the board assumes the administrative burdens of the investigation. In addition, the actual occurrence of mishaps

in aviation activities is not routine and thus administrative procedures may be the results of crisis management rather than established procedure. Although safety directives require the commanding officer and the ASO to conduct training and drilling, they cannot predict the workload associated with the actual mishap occurrence.

D. SOLUTION

Our proposed solution to the problem is the implementation of a system which addresses the above problems of uncertainty, time constraints and administrative overhead. This thesis addresses the entire mishap investigation process with a focus on the central issue of the mishap investigation, the deliberation of the board. We concentrate on the development of a "deliberation model" because it directly addresses the purpose of the Mishap Investigation Report, that of accurately identifying hazards or causal elements. We provide an example implementation of this model to demonstrate its usefulness in a simulated mishap. In addition to the development of the deliberation model which is the centerpiece of the DSS, we suggest schemas for the development of the related databases supporting the "deliberation engine". It is the role of these supporting functions which will complete a DSS development by automating the report generation processes. In addition, we seek to automate many of the decisions involving mishap classification and rules in reporting defined by the Naval Safety Center Program instruction

This thesis presents a decision support model which is logically based and seeks to aid the investigator in the investigative process. The model does not strive to automate the decision making process, rather to introduce efficiencies made available by principled information processing and storage. In addition, the model presents a context which is visually based, and although the implementation presented in this thesis does not provide a visual tool, the basis for the model is best represented in this graphical context. The thesis examines in detail the problems of mishap investigation and focuses on the core of the proposed decision support system. Finally, we present a basic implementation using a "canned" mishap investigation and propose further development.

II. DETAILED DESCRIPTION OF THE PROBLEM

A. THE NAVAL AVIATION SAFETY PROGRAM

OPNAV instruction 3750.6Q is the current version of the instruction governing the mishap investigation process. The mandate of this instruction is much broader, however and includes most aspects of the Naval Aviation Safety Program including the conduct of individual aviation command safety programs. This thesis enhances the purpose of the Naval Aviation Safety Program by improving the mishap investigation process as defined in the 3750.6Q instruction. The instruction states this purpose on page 1-1:

"The purpose of the Naval Aviation Safety Program is to preserve human and material resources. The program enhances operational readiness by preserving the resources used in accomplishing naval aviation missions. The human resources include professional pride, high morale, physical well-being, and life itself, all of which are susceptible to damage and destruction by mishaps. The material resources include all kinds of property which might be damaged by a naval aircraft mishap, such as naval aircraft, ships, weapons, and facilities. The Naval Aviation Safety Program thus directly supports all aspects of naval aviation. Resources other than naval aviation resources may be preserved through success of the program, and knowledge gained in the program may assist other safety efforts. The program, therefore, yields benefits beyond its scope."

The instruction further defines the objective of the program on page 1-1:

"The purpose of the Naval Aviation Safety program is accomplished by the prevention of damage and injury. Potential causes of damage and injury are termed hazards. The objective of the Naval Aviation Safety Program is to eliminate hazards."

The Program seeks to eliminate hazards through their identification, and mishap investigation is one of the primary vehicles for this identification. In addition to the understanding of the Program and Objectives, this thesis utilizes some basic concepts and definitions reviewed in section 105 of the instruction. We base much of the deliberative model on the following concepts defined in the instruction:

- ♦ *Necessitarianism*: The doctrine that events are inevitably determined by preceding causes, and the corollary that events may be prevented by the elimination of their causes.
- ♦ *Damage and Injury* are the events to be eliminated and thus the causes of damage and injury may be prevented by eliminating their causes.

The causes of damage and injury are hazards and thus the purpose of the program is to eliminate hazards. Logically, the ideal outcome of the Naval Aviation Safety Program would be the elimination of all hazards or causes of mishaps. By this definition, a mishap is a failure of the Naval Aviation Safety Program. The instruction describes the importance of the sequences of prescribed action necessary both before and after the occurrence of a mishap; when these actions occur after the mishap, their purpose becomes that of preventing a recurrence. The instruction defines these actions specifically on page 1-3 as "hazard detection" and "hazard elimination".

Hazard detection and elimination after a mishap are the responsibility of the permanent Aircraft Mishap Board (AMB) appointed by the reporting custodian of the aircraft involved. The purpose of this board is specifically to "detect hazards through mishap investigation" [OPNAVINST 3750.6Q]. The instruction further states provisions for hazard reporting after the mishap in the form of the Mishap Investigation Report (MIR) which is required for all defined naval aircraft mishaps. The MIR is the final product of the AMB and serves the primary purpose of hazard identification. In addition to this identification, the AMB provides recommendations for the elimination of the identified hazards through formal recommendations of corrective actions which require documented action. Thus the AMB becomes a powerful force in the elimination of identified hazards after their identification.

B. PROGRAM PARTICIPANTS

1. The Senior Member

The investigation, deliberation and formulation of the Mishap Investigation Report fall under the responsibility of the Aviation Mishap Board (AMB). The Naval Aviation

Safety Program instruction requires reporting custodians (usually commanding officers of aviation squadrons) to appoint and maintain a standing AMB. Technically, the reporting custodian must also appoint an AMB "senior member" to take responsibility for the training and readiness of the AMB but the senior member is usually the reporting custodian (also the commanding officer). The Senior Member acts as the "chairman" of the board and is ultimately responsible for a given mishap investigation.

2. The Aviation Safety Officer

The Aviation Safety Officer (ASO) is a designated naval aviator or naval flight officer who is a graduate of the "Aviation Safety School". The ASO is a participating pilot or flight officer in the primary mission of the squadron and possesses additional training as a specialist in the Naval Aviation Safety Program from the Naval Aviation Safety School at the Naval Postgraduate School. The ASO school also provides the ASO with specialized training in such areas as wreckage examination, structural engineering, human factors in mishap investigations, and the conduct of mishap investigation and reporting. Although the training and maintenance of the AMB is the responsibility of the Senior Member of the AMB, since the Senior Member is usually also the commanding officer, these responsibilities are typically delegated to the ASO.

In the context of an actual mishap, the ASO is the primary functionary in the investigation process. The ASO possesses the basic knowledge and acts as the translator between the technical experts and the AMB members. The ASO's task is similar to that of a lawyer in a courtroom, he must take highly technical data and evidence from expert investigators and translate it to the level of expertise of the AMB. This often requires that the ASO provide training for the AMB in certain areas to facilitate clear understanding of the evidence. In addition, the ASO is responsible for the accurate collection and interpretation of evidence by technical sources. The ASO school training seeks to prepare ASOs for this by providing a level of training which will make the ASO at least conversant in the technical areas encountered in mishap investigations. Experts such as Naval Safety Center investigators and design engineers do not participate in the deliberations of the

AMB in a mishap, they only provide evidence on which the AMB deliberates. It remains the responsibility of the ASO to provide the evidence to the appropriate experts and then return them for AMB deliberations.

3. The Aviation Mishap Board

The Naval Aviation Safety Program governing instruction requires that each "reporting custodian" maintain a standing Aviation Mishap Board. Paragraph 206 (pp.2-4 to 2-6) states that the members shall be appointed by name and in writing by the designated appointing authority. The AMB is basically composed of active duty, commissioned officers of the USN or USMC. At a minimum the board must include four officers including a Senior Member, an ASO, a flight surgeon, an officer well qualified in aircraft maintenance and an officer well qualified in aircraft operations. If necessary, the appointing authority may designate members from outside the command if experienced or qualified officers are not available within the command. In addition, for mishaps involving aircraft manned by aircrew, the board shall consist of at least one officer who is qualified in that particular model of aircraft. Typically, squadrons or aviation activities maintain mishap boards composed of four to ten officers, and include designated alternate members to replace primary members who might not be available at the time of a mishap.

The Senior Member of the AMB ensures (through the ASO) that the board is trained and prepared for a Mishap. This training is formalized in A "pre-mishap" plan which serves as a contingency plan for implementation in the event of a mishap. The plan serves as the training document for the AMB and anticipates "all reasonable eventualities" encountered in the aftermath of a mishap and attempts to cope with these eventualities. Site security, media coordination, area law enforcement, and wreckage preservation are just a few of the issues addressed in the pre-mishap plan. A decision support system implementation would necessarily become part of this plan: And the Pre-Mishap plan should address hardware, software and connectivity as appropriate to the computing environment on which the system is implemented.

4. Other Experts

Section 603f and section 608 of OPNAVINST 3750.6Q describe sources of assistance outside the resident AMB which a Senior Member may request during the course of an investigation. The Chief of Naval Operations may mandate a Naval Safety Center investigation, or the AMB may request investigative assistance. The Naval Safety Center provides professional investigators to the AMB and serves as another source of evidence when requested or mandated. This investigative assistance does not absolve the AMB of responsibility, it only enhances the ability of the board in high- profile or difficult investigations. In addition to investigative assistance, an AMB may request technical and medical assistance from other defense agencies. The sources of this assistance are varied, they range from engineering assistance from government laboratories to forensic pathology evaluations from the Armed Forces Pathology Institute. Since there is often a "knowledge gap" between the expert assistance and the AMB knowledge base, ASOs are trained in the more common areas to understand the technical results and translate them for the AMB. The results of expert assistance are advisory and the AMB deliberations treat all assistance as evidence.

C. MISHAP INVESTIGATION

In designing a decision support system for aviation mishap investigation, the task of this thesis is simplified by the highly structured requirements of the Naval Aviation Safety Program. In seeking to identify the cause factors in a mishap the program requires the final product, or the Mishap Investigation Report be in a rigid format which requires logical support for arguments. The program instruction does not however, provide guidelines for detailed deliberation and decision making techniques beyond the formatting of the final product. The Safety Program does not provide guidance for the investigation process. The Senior Member, the ASO and the members of the AMB are left to determine a strategy for evidence collection, cataloging and deliberation. This thesis takes advantage of the MIR structure to develop the deliberation model.

The deliberation model developed in this thesis does not require a detailed description of naval mishap classes. As background however, the reader should understand that mishap classes are categorized by severity based on monetary losses and human casualties, and also by types of occurrences such as flight mishaps and ground mishaps. The most severe mishaps are class "A" and require the most stringent reporting requirements. Class A mishaps require the AMB to respond more quickly with an initial report and subject the final report to detailed review not required of lower classifications of mishaps. Our model development targets the environment encountered in the investigation and reporting of a Class A mishap but is germane to other investigations as well. The basic elements of the model are common to all naval aviation mishap and hazard investigations.

1. The Mishap Investigation Report

The Mishap Investigation Report (MIR) is the result of the deliberations of the Aviation Mishap Board following a naval aviation mishap. The purpose of the MIR is to report and document the hazards which were the cause of the mishap and damage and injury which may have occurred in the course of the mishap (sec 702). The MIR develops a list of evidences into "detailed cause factors" through a series of logical steps, and these cause factors are the end result of the mishap investigation. Once the MIR is completed, the AMB publishes it as a naval message. As the message passes "up the chain of command", each endorser comments on the investigation and addresses specific recommendations made by the board. A more detailed description of the process is represented in figure 2-1.

a. Evidence

After the occurrence of a mishap, an initial message is published and the formal investigation convenes the AMB. The AMB eventually collects the evidence, deliberates and then publishes the MIR. The first significant section of the MIR is the evidence section. The investigative process results in a list of evidences which are categorized in the MIR in a summary format. Each piece of evidence is part of an

"enclosure" which denotes the location of the described physical evidence. The evidence section thus contains the factual data in the mishap describing all of the relevant evidence

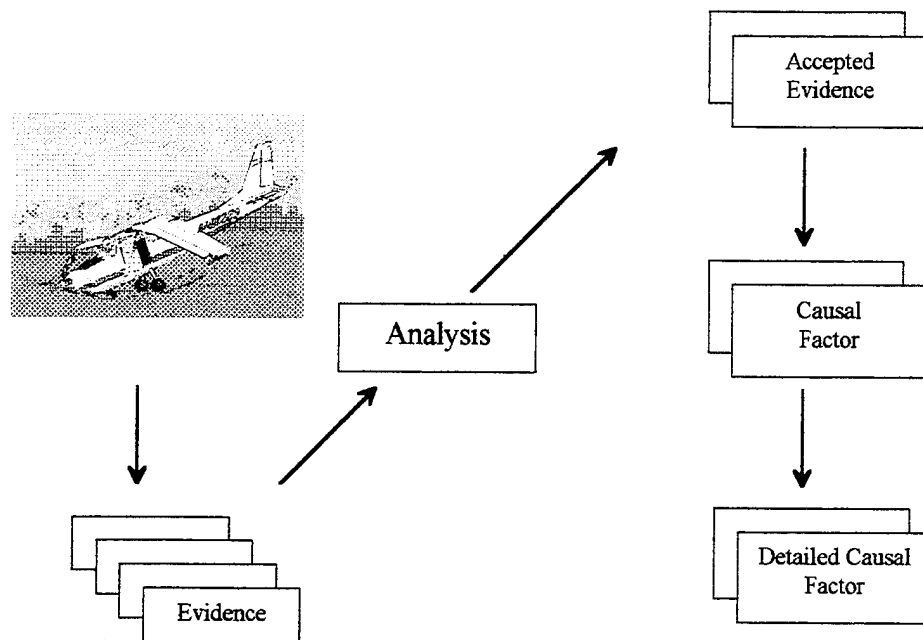


Figure 2-1 Mishap Information Flow

in the investigation from all sources.

b. Analysis

The next section of the MIR contains the analysis paragraph. The AMB documents the investigation by separating the mishap into "factors" which it either accepts or rejects. The given factors are described as aircrew, supervisory, facilities, maintenance or material and must be based on the evidence included in the previous section. Section 716 of the Safety Program instruction describes each rejected or accepted factor as a "scenario" which is tested by the board in light of the evidence. This section includes a summary of each scenario by describing the deliberations of the AMB in

reaching their conclusions. The instruction suggests that a useful technique for describing these factors is in chronological sequence when documenting deliberations.

c. Conclusions

The "conclusions" section of the MIR classifies the accepted factors from the analysis section and determines their individual severity using "risk assessment codes". In addition, the AMB must further classify each accepted factor as a "detailed cause factor" which is the finest level of classification in the investigation. The AMB must translate each accepted factor to a "detailed cause factor" provided in appendix L of OPNAVINST 3750.6Q. This list is "an exhaustive tabulation of the way in which people and aviation machines have historically interacted to produce mishaps; as such they provide a menu of possible Human Factors that could be involved in a mishap" [OPNAVINST 3750.6Q p.L-1]. These cause factors are divided into *who*, *what* and *why* or *component*, *mode* and *agent* categories and serve to contribute to the Naval Safety Center's mishap data file. The AMB chooses the "Who/What/Why" tabulated cause factors most appropriate to each developed factor accepted in the analysis, and this set of factors represents the final outcome of the mishap investigation process. A separate section of the MIR includes causal factors causing other damage or injury in the mishap which are indirect results of the mishap. This section uses a slight variant of the process described above.

D. THE INVESTIGATION PROBLEM

1. Example Presentation

OPNAV Instruction 3750.6Q provides the following fictitious example to present message formatting:

"Scenario: GEAR-UP LANDING

A multi-piloted aircraft joined the landing pattern. The aircrew consisted of pilot (aircraft commander), and copilot (pilot qualified in model). The copilot, a brand new nugget recently reported, read the landing checklist and the pilot, a seasoned veteran of intimidating

demeanor, executed it. The pilot put the landing gear handle in the down position but did not check the gear position indicators. They showed the gear up, and neither pilot noticed the gear warning light which was illuminated. The gear was, in fact, up. That particular aircraft was equipped with a warning horn which sounded when the throttle was retarded to a descent setting and the landing gear was up. The horn failed to sound when the pilot retarded the throttle at the 180. The aircraft subsequently landed gear-up with Class "B" damage.

The following facts were discovered in the ensuing investigation: The pilot had only four hours sleep the previous night because he worked late; the pilot's father had died the month before; maintenance work done previous to the flight had been in accordance with directives but a significant step was omitted from the maintenance handbook which allowed the gear handle to be lowered without lowering the gear; an emergency backup to lower the gear was available but not used; a microswitch inside the throttle quadrant was found corroded and failed as an open circuit, the microswitch activated the warning horn system; the normal climate at homebase at that time of year was wet and rainy, the aircraft had not been hangared on a regular basis which was not required." [3750.6Q, p. M-1]

The purpose of the above example mishap is to illustrate the process of taking evidence discovered in the course of the mishap investigation to compose paragraphs 10 through 13 of the MIR [3750.6Q, p. M-1]. The authors of the instruction admit the example is hypothetical, brief and contains many logical errors, and thus serves this discussion well in pointing out the utility of the model. The "facts" or evidence discovered during this example investigation are incomplete and useful for illustrating the deliberation model. The discussion will argue the utility of the deliberation model based on this example, and draw additional inferences which might aid the fictitious Aviation Mishap Board investigating this mishap. During the subsequent discussion, we will repeatedly refer directly to this example to enhance selected themes.

2. Why is there the Need for an Automated Deliberation Process?

The focus of this thesis is the development of the deliberation model. The presentation of the underlying, formalized principles provides a reference for future implementations of decision support systems using this simple logic with advancing

technology. Automation of the deliberation process addresses the "macro" problems in Chapter 1, and we can specifically state influences on the deliberation process likely to affect the deliberation process. Three of these are "selective inattention", bias and logical errors encountered in deliberations.

a. Selective Inattention

Selective inattention or cognitive filtering acts to "hide" evidence or probable causes based on any number of influences. [Frew, 1995.] The unconscious act of selectively omitting or discarding relevant information in any context leads to poor decision making, or in this case, inaccurate deliberation. Selective inattention may be the result of the opinions of an influential AMB member, or it may result simply from various pressures put collectively on an AMB or its members. For example, squadron loyalty and pride may make the deliberations of an AMB difficult if the reputation of a popular commander is marred by results. As the AMB deliberates, they might *selectively* and unconsciously omit any chain of events which may point to the commander. The organized approach presented by the deliberation model forces a continuous process of inquiry which, if properly implemented, will address selective inattention.

b. Bias

If the AMB deliberates an investigation in an unstructured manner, the possibility of deliberate bias becomes much greater. Although the concept of "privilege" [OPNAV 3750.6Q p.1-5] protects witnesses who provide evidence in a mishap, this concept cannot fully protect the findings of a board. Deliberations which result in politically sensitive or volatile conclusions can face external, deliberate pressures which affect board members consciously and unconsciously. In the extreme, a biased member can deliberately try to direct or sway deliberations, and the Senior Member might not detect this if he does not organize and document the deliberation process. Biased conclusions will not develop logically, and credible evidence will not support them. This deliberation model presents a causal flow and evidence assignment which exposes such bias.

c. Logical Errors

A logical approach to mishap investigation is not difficult, and yet logical errors cause AMBs to re-deliberate and re-visit issues unnecessarily. Unfortunately the reporting custodians of aircraft involved in a mishap (commanding officers) who release flawed investigations receive immediate feedback from Senior's endorsements, making their logical errors very public. The model presented here assumes the AMB commits logical errors when they fail to adequately support a conclusion based on a probable cause, or inadequately support a probable cause with evidence. In addition, the AMB can commit logical errors by not completing the investigation, i. e. the board fails to seek evidence to support the most "detailed" probable cause at the end of a "chain of events". The quoted example provides an illustration of this type of error: Although the board discovered that the pilot had only four hours sleep the previous night, the board failed to continue the investigation and consider why he was awake. This approach might force the board to investigate the possibility of a physiological or psychological problem the pilot suffered rather than accepting the lack of sleep as the root cause.

In this chapter, we have presented the detailed description of the problem of mishap investigation, focusing on the deliberation process (or lack thereof). We have attempted to illustrate some, but not all of the problems associated with an unorganized approach to the deliberation of the AMB. Current directives leave the deliberation process open to interpretation and yet this process should be the basic building block of any mishap investigation. This theme is recognized by the professional instruction staff at the Aviation Safety School and this staff attempts to address this lack of methodology in the instruction given to Aviation Safety Officers. This instruction serves as a basis for the subsequent model development.

The deliberation model is the center of the thesis, but it is supported by other modules of the entire system. These modules seek to address the problems discussed in Chapter 1, and their implementation is discussed in the following chapter in advance of the detailed deliberation model development. The discussion of the system development lays

the foundation and provides many of the assumptions germane to the deliberation module and model discussions which follow in the subsequent chapter.

III. A DECISION SUPPORT SYSTEM FOR NAVAL MISHAP INVESTIGATION

A. SYSTEM OVERVIEW

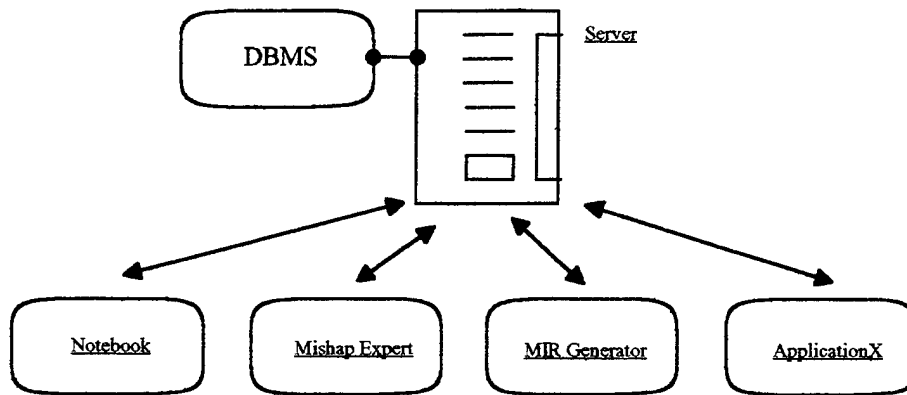


Figure 3-1 Application Architecture

1. General description and requirements

Figure 3-1 describes the overall proposal for the complete decision support system. Developers may follow this architecture in a single application treating each separate box as a module, but the client/ server approach best satisfies the requirements of this application. Specifically, the above proposal suggests "database server" type architecture commonly referred to in client/ server terminology [Orfali, Harkey and Edwards, 1995]. The requirements of the system are basic. They simply automate the mishap reporting process. In addition, the system shall aid the AMB in the deliberation process. The client/ server architecture facilitates the standardization of data, the portability of individual component applications and data, and the ability to develop and easily implement additional applications based on future requirements. In addition, the intelligent and careful development of the server database might provide interface with other activity functional databases such as personnel or maintenance departments.

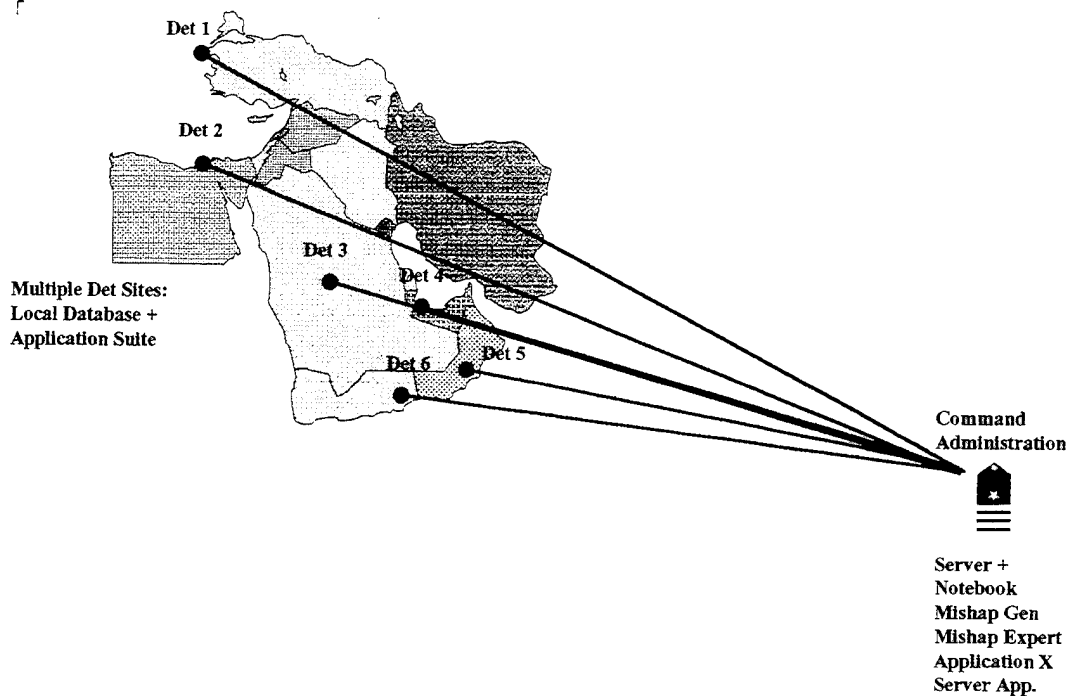


Figure 3-2 Distributed Environment

A stand-alone or single platform client/ server implementation similar to Figure 3-1 utilizes a single database. Implementations which involve distributed computing environments such as the "detachment scenario" described in Figure 3-2 might require a slightly different architecture in which remote client applications communicate with a command server. Often, naval aviation squadrons or activities are tasked with operations over a large geographical area, necessitating the establishment of temporary "detachment" sites which serve as sub- commands of a parent. In these situations, the parent command may delegate many of the reporting and investigative tasks to the detachment commander, but the message release authority and central administration functions reside at the central command location. Necessarily, developers would attach a maintainable, local database capability at the detachment site which would communicate with the command server system for database updates and mishap investigation administration such as revision and modification of mishap investigation messages. This type of client/ server architecture is

described by a distributed data management model which allows the client application to manage the presentation, application logic and data management, and tasks the server with data management only [Orfali, Harkey and Edwards, 1994]. If the command location is also an operational site, then the command will become a client and also maintain the server and associated DBMS. This cooperative processing allows the generation of the final investigation product at any detachment or client site while the server platform provides the squadron- wide database services.

B. MODULE OR APPLICATION DEVELOPMENT

1. Server and Database Design

The OPNAV 3750.6Q instruction provides a reference for database development. A careful examination of the database model presented in Appendix A will illustrate our strategy of providing a reusable database while replicating the data elements required by our deliberation application presented in Chapter IV, and other specific elements required by the system we propose in this chapter. In addition to this semantic table, the ANSI SQL-92 Schema is included in Appendix B. An examination of the mishap and mishap investigation message formats suggests some of these data objects such as Evidence, Factors and Cause Factors. This thesis follows the semantic object database design approach and we develop the schema based on the semantic object modeling in Appendix B.

The development of the above schema seeks to provide data elements common to a mishap with reuse by other squadron activities in mind. For example, a maintenance department data application might make use of the "Aircraft" object, or an administrative office might utilize the "person" object. We developed this particular schema to support the applications treated in this thesis, however, and future developers may consider modifications appropriate.

2. Initial Reporting

The initial reporting application should provide the timely release of the initial mishap message and voice reports according to the limits specified in the OPNAV 3750 series. This application will also set the default information for a particular mishap and automate the decisions associated with the classification of a mishap. To produce the initial mishap message and voice report alone, the message drafter must make numerous significant decisions requiring the reference of the OPNAV 3750 instruction:

- The existence of a "defined" naval mishap,
- The category of the mishap as described by paragraph 401 of OPNAV 3750,
- The severity of the mishap as described by paragraph 413.

Specifically, these decisions are formally represented in the decision trees located in the Appendix A of Chapter IV of the instruction and provide developers with a template for the decision code generation. Readers should review this instruction and the associated decision trees for further information.

An excellent example of an implementation of the decision logic of the 3750 instruction was developed by LT Hugh Brian while assigned at the Naval Aviation Safety School in June of 1995. His implementation was accomplished using the Delphi Rapid Application Development tool and the language used is ObjectPascal. We present this implementation as an excellent example of an application supporting the initial reporting phase of a mishap. Although the application does not support the client server architecture described here, the interface and decision mechanisms provide templates for future development.

The application "Notebook" provides the user with a number of important features. Brian allows the user to set default values which are locally preserved for future use. Among these values are "Officer Recall", and default squadron information. The following figures present the screens developed by LT Brian in the Mishap Notebook application. In addition, the application contains a built- in template for the initial message generation and the capability to store that information as a text or message- formatted file.

The application also provides the user with the capability to generate an "OPREP" message (a naval incident reporting message) and an "OPREP" phone report. The interface of the application guides the user through the process of the initial mishap message reporting procedure (Figures 3-2 and 3-3), using a checklist to ensure the user

Figure 3-3 Notebook Initial Info. Screen

Officer Recall	Rank/Name	Squadron	Home Phone	DNS Work Phone
CDR/C	NAME		COMPHONE	AUTOVON#
XO	NAME		COMPHONE	AUTOVON#
OPS Officer	NAME		COMPHONE	AUTOVON#
Maint. Officer	NAME		COMPHONE	AUTOVON#
Flight Surgeon	NAME		COMPHONE	AUTOVON#

Proposed Mishap Board	Rank/Name	Squadron	Home Phone	DNS Work Phone
A&MB Senior Member			COMPHONE	AUTOVON#
Flight Surgeon	NAME		COMPHONE	AUTOVON#
Maint. Officer	NAME		COMPHONE	AUTOVON#

Figure 3-4 Notebook Officer Recall Screen

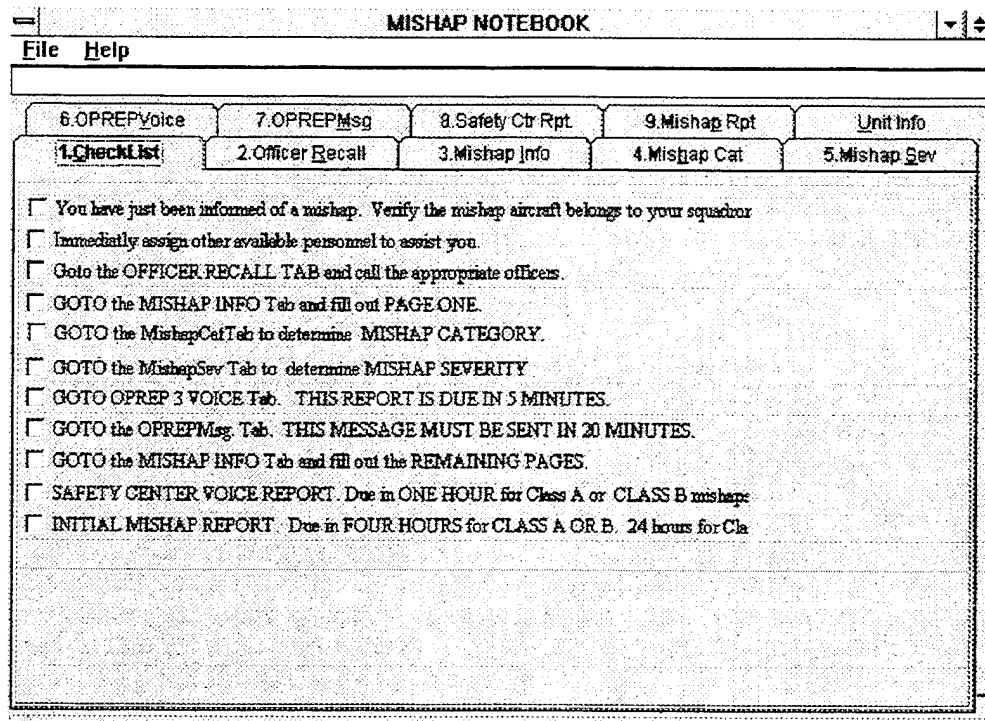


Figure 3-5 Notebook Checklist Screen

complete all the required items. Figure 3-4 is the checklist screen implemented in the application. This interface allows the user to gage his progress and also informs him/her of the elapsed time with a running clock, which continues to remind the drafter of the time requirements.

a. The Mishap Category Decision

Notebook aids the user in selecting the mishap category with the interface presented in Figure 3-5, the "Mishap Cat." screen. The OPNAV 3750 instruction provides a "decision matrix" to determine the mishap category based on nine different conditions. In order to aid the user in selecting the correct mishap classification, Brian presents the user with a series of standard windows check boxes in a interview format. The user's responses are recorded and the program executes the following Pascal code to determine the correct category of mishap:

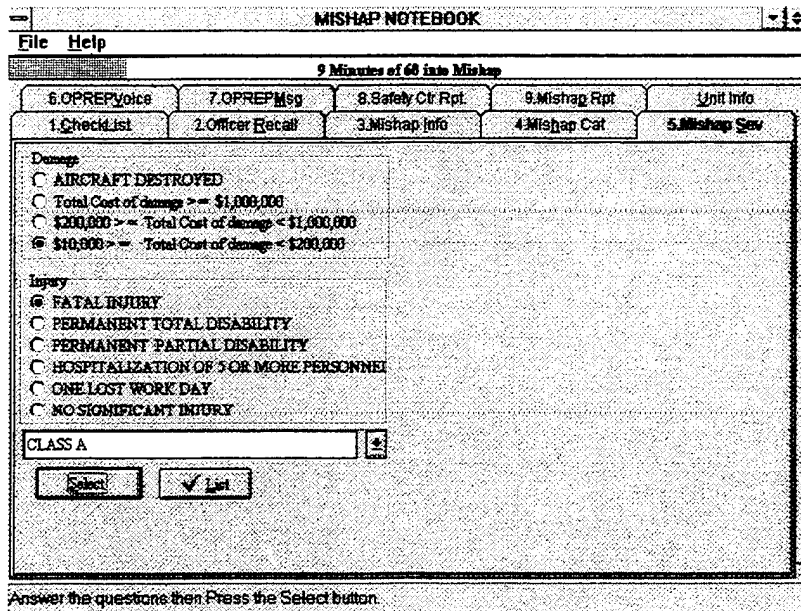


Figure 3-6 Notebook Severity Screen

```

procedure TNoteBookForm.MishapCatDoneClick(Sender: TObject);
var
    MishapType, MishapInjury:TMishapClass;
begin
    if (Damage.ItemIndex = 0) or (Damage.ItemIndex = 1)
    then MishapType:= ClassA;
    if (Damage.ItemIndex = 2) then MishapType:=ClassB;
    if (Damage.ItemIndex = 3) then MishapType:=ClassC;

    if (Injury.ItemIndex = 0) or (Injury.ItemIndex = 1)
    then MishapInjury:= ClassA;
    if (Injury.ItemIndex = 2) or (Injury.ItemIndex = 3)
    then MishapInjury:=ClassB;
    if (Injury.ItemIndex = 4) then MishapInjury:=ClassC;

    if (MishapType = ClassA) or (MishapInjury = ClassA)
    then MishapSeverity.ItemIndex:= 0 else

```

```
    if (MishapType = ClassB) or (MishapInjury = ClassB)
    then MishapSeverity.ItemIndex:= 1 else
    if (MishapType = ClassC) or (MishapInjury = ClassC)
    then MishapSeverity.ItemIndex:= 2;
end;
```

The above code executes the decision trees located OPNAVINST 3750.6Q. Specifically, when combined with the screen which determines the "intent for flight", this code executes the decision tree on page 4C-1 of the above instruction titled "Severity Classification". When the user defines and selects certain exclusive check boxes, the simple series of Pascal "IF- THEN" statements select the correct category when the button is selected. Figure 3-5 is the screen implementation of the above ObjectPascal code. The ".itemIndex" extensions indicate the objects which contain the values of the checked box, and these returned values determine the classification of the mishap.

LT Brian's application is currently a stand-alone version which saves information only to a text or a message file. It is presented here as an example of a successful implementation of the decision matrices specified by the Aviation Safety Reporting instruction. In order to be applicable to this system, however, developers would need to rewrite this application to include database access. The Delphi software package and its object oriented approach to programming will facilitate conversion of this type application to include database functionality. In doing so, developers should follow the server schema presented below when considering the local database design. At this thesis writing, version 1.1B of the entire system is under development at NPS attempting to convert LT Brian's application into a "client" application in our framework (with the permission of the author). In implementation of this system, we strive to implement LT Brian's interface and design, rewriting the code to communicate with the server database. We also would preserve the additional capabilities included in the Mishap Notebook application as part of the entire system. LT Brian's Mishap Notebook application can be obtained on the internet at the Naval Postgraduate School Aviation Safety homepage

(<http://www.nps.navy.mil>) or from the Aviation Safety FTP site located at "nps.navy.mil" on the internet.

3. The Final Mishap Reporting Application

Developers may include the final reporting application with the AMB deliberation application presented in the next chapter. If implemented separately, however, the Final Mishap Reporting Application will serve to produce the final mishap message in accordance with OPNAV 3750.6. The database schema suggested for the server facilitates the reuse of the original message by simply appending the latest version of the initial message. The referenced instruction specifies the first nine paragraphs as those appearing in the initial message and the subsequent sections specifically treat the reporting of the deliberation and investigation of the mishap after the initial report.

Currently, version 1.1B of the Decision Support System in development during the writing of this thesis supports a separate Mishap generation application which imports specific data from the AMB deliberation application module. Once this information is processed in the final message generation application, the user application should prepare the final message. We do not address this portion of the system in detail because it is a simple application which developers may easily implement.

This application should provide a template function which maintainers can easily update and revise. Ideally, this "template updating" function should reside in a server application, so the client applications can receive new templates when needed. A system administration type function should be integrated into the suite of application included with the server maintenance to update templates and provide them to client applications.

The following chapter presents the central focus of this thesis, the deliberation application.

IV. THE DELIBERATION APPLICATION

A. BACKGROUND

1. The Aviation Safety School Model

The central issue addressed in this thesis is the development of the deliberation model and a supporting application. This application or functionality seeks to improve the quality of the deliberation process during mishap investigation by providing an interaction with the user which will record and organize the reasoning of the AMB. The goal is not to completely automate the decision process, rather to present the user with an organized and proven strategy for effective presentation of the evidence, causes and conclusions of a mishap deliberation.

The genesis of the model we develop below is a version of "blackboard" method currently taught by Aviation Safety School faculty to Aviation Safety Officer trainees in

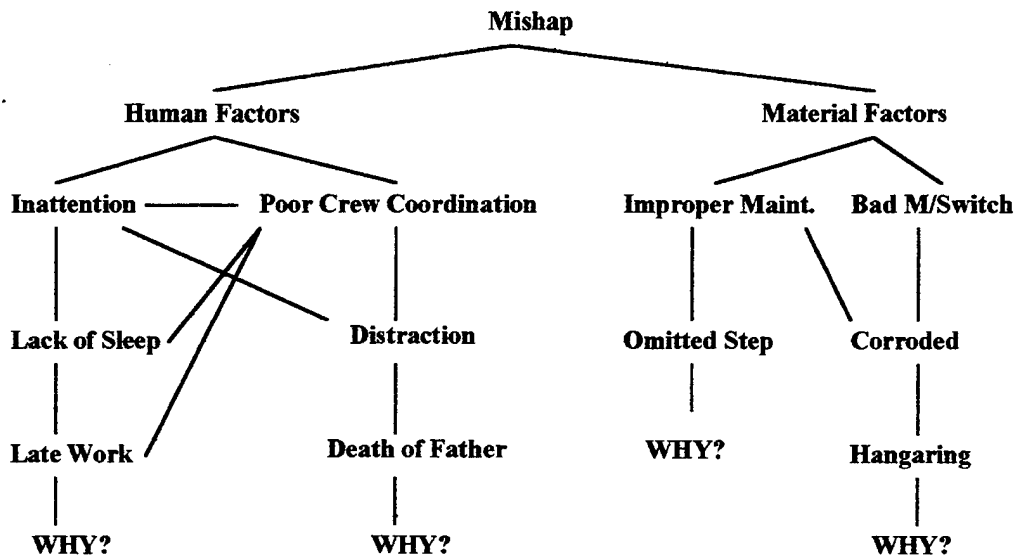


Figure 4-1 Aviation Safety School Mishap Investigation Model
(Maj. Tom Hazard)

the Mishap Investigation course. Major Tom Hazard, USMC an instructor in Mishap Investigation, uses the model reproduced from his teaching notes in Figure 4-1. This version of his model includes the data from the example mishap in Chapter I.

Maj. Hazard's model characterizes the methodology we follow in the development of a similar model below. The model should accomplish the following goals through interaction with the agent:

1. It should present the evidence and facts in the form of a logical "chain of events".
2. It should allow the investigator to infer differing influences on these events.
3. It should prompt the investigator to seek evidence supporting the "ends" of these chains by continually researching the simple question "why?" using logical deduction.
4. It should allow the AMB to "test" various scenarios using logical abduction.
5. It should provide a useful interaction with the investigator to "visualize" the mishap.

When we speak of a logical chain of events in the first point, we refer to a logical placement of causal elements rather than a temporal one. The temporal chain of events focuses on the order of events rather than the reason for their occurrence. Using a logical chain of events, often a temporal order occurs, but the reasoning is more organized. For example, an investigator in our "toy" mishap would continue to ask why an event occurred until he produced evidence and then would ask the same question again. The end result of deliberation is a series or a set of statements about the mishap, the statements are the result of dialog among the members of an Aviation Mishap Board. The following is a sample logical dialog which might occur during a mishap board deliberation:

Why did the mishap occur? Because the gear was up. This is supported by the wreckage photographs.

Why was the gear up? We think it is because a microswitch failed. This is supported by a wreckage photograph which shows the switch in the open position.

Why did the switch fail? We suspect that the switch failed because of corrosion we observed on the switch. This is supported by the results of an engineering investigation stating that there was corrosion on the switch.

Why was there corrosion on the switch? There is a possibility that the switch was poorly manufactured, but there is no evidence to support this (or there might be evidence to refute this).

Why was there corrosion on the switch? We think that the aircraft was improperly hangared and corrosion resulted. This is supported by maintenance records that reflected a violation of hangaring regulations.

Why were the regulations not followed? We think poor supervision is the result of the improper hangaring and this is supported by maintenance records.

The above discussion follows the model suggested at the Aviation Safety School. This logical connecting of true statements forces the investigator to examine each scenario in a detailed manner, and supported arguments become the basis for the continuation of the logical chain and investigation. The difficulty with the logic, however, lies in the complexity of naval mishaps. Mishaps occur with varying amounts of evidence and AMBs must consider numerous scenarios. An automated system supporting this type of logical process serves to record and document the reasoning process used during an AMB deliberation session and also serves to export the results of the deliberation directly into the final Mishap Investigation Report product.

B. THE AMB DELIBERATION MODEL

We seek to automate our version of the Safety School model by formalizing the model and providing examples of research which supports the use of such a type of reasoning in modern decision theory. The type of diagramming used by Maj. Hazard is not uncommon to professional disciplines and is a type of human cognition known as "cognitive mapping" [Eden, 1988],[Duncan and Paradise,1992]. Cognitive mapping is broadly applied to many different disciplines but in this case refers to the "representation perceived by an human being to exist in a visible or conceptual world" [Zhang, 1994],

[Axelrod, 1976]. In Mishap investigation, the problem space is the entire mishap event consisting of a set of evidences and causes. Ultimately, the cognitive map will look much like Maj. Hazards diagram, but we provide a context in which we can apply formal logic and Artificial Intelligence methodology.

Academicians in all fields have recognized the importance of cognitive mapping for almost a century, and recently it is becoming more important as computer graphics and Artificial Intelligence provide a means to manipulate the data in a cognitive map [Axelrod, 1976]. Not only can we reproduce the graphical representation of a mishap investigation cognitive map, but we can potentially automate the entire process. With enough data and tools such as neural networks, we can envision systems which analyze mishaps and determine the most likely causes independent of human decision processes. Although these capabilities exist, the practical and political implications of a system which excludes human reasoning would make implementation impractical in the current aviation safety environment. We propose to find "middle ground" between the blackboard method of cognitive mapping and the total reliance on computer technology in the investigation process with a system which allows the user to reason using the computer as a guide and administrator.

When visualizing the mishap event in the manner taught by the Aviation Safety School, we create a cognitive map. The effectiveness of the cognitive map concept is proven in numerous artificial intelligence applications and can serve as a basis for model development. The importance of cognitive mapping is illustrated by a study conducted by Rook and Donnell which suggests that the most powerful explanation format in expert systems is the graphic-based inference explanations based on existing user problem spaces [Rook and Donnell, 1993]. In our context, the existence of visual models such as the one developed by Hazard provide an cognitive map which we should seek to support in the graphical context. We believe it is reasonable to assume that a cognitive map of the mishap problem space is a common visualization in mishap investigation (although we can only informally verify this) among investigators. Although we cannot assume that a

specific type of cognitive map will be perfectly understood by an AMB, the representation presented at the Aviation Safety School provides a good basis. In designing the model and the interface, we should ensure that the interface communicates the graphical problem space and the textual rule based problem space to the user to elicit the greatest user/interaction performance as Rook and Donnell suggest.

In order to formalize the Deliberation Model, we apply the cognitive mapping concepts to a formal knowledge representation scheme. The concepts of semantic networks in artificial intelligence combine the use of textual, graphical and visual representations. In using a semantic network construct, we combine the idea of cognitive mapping with an artificial intelligence heuristic which we can easily formalize. The original development of semantic networks was defined by Quillan and utilizes a set of nodes and arcs or edges to represent the problem space [Quillan, 1968]. The significance of the nodes and arcs vary with the goal of the model, but the use of semantic networks is widespread. Quillan's original heuristic, for example, defined the English language through detailed definitions of nodes and arcs. Another type of network was developed by Reiger [Reiger, C., and M. Grinberg, 1977], and uses nodes and arcs to define causal relationships. Reiger's causal networks facilitate descriptions of such things as a the operation of a reverse- trap toilet [Gonzalez and Denkel, 1993]. Semantic networks provide a great deal of flexibility in describing a problem space, and facilitate reasoning about the problem. The semantic network we present is an extension of Major Hazard's work and allows us to take advantage of an easily understood problem space.

1. Model Description

We describe our model formally in the context of a directed network composed of the elements mentioned in the paragraph above. This semantic network is "directed" because it defines relationships from a "root" probable cause or evidence to a "leaf" probable cause or evidence. We clarify the definitions of these terms below. Figures 4-2 and 4-3 illustrate examples of the network, using our previously described toy mishap. Figure 4-2 is a branch of our semantic network, indicating its basic components. Similar

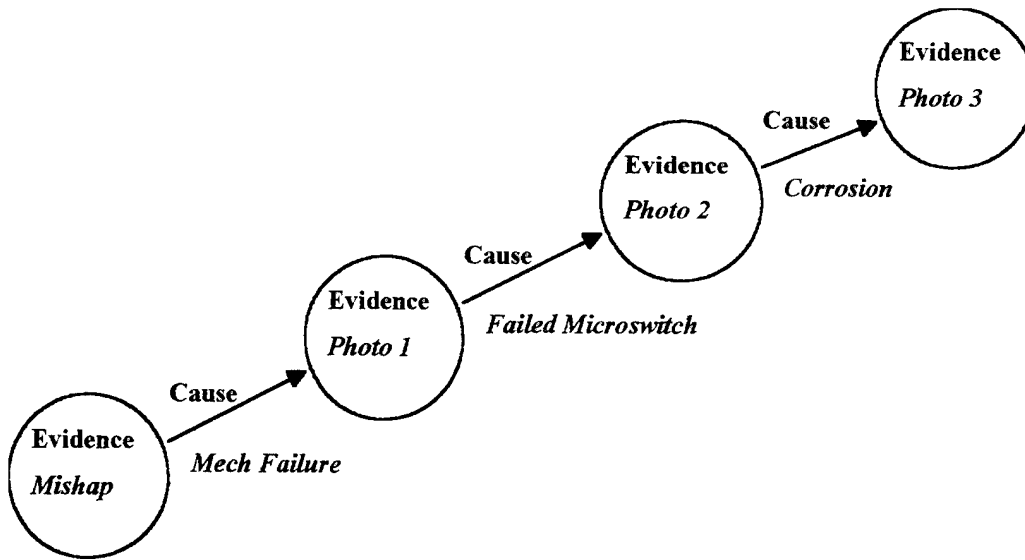


Figure 4-2 Deliberation Model Branch

to Major Hazard's model, the network expands outward with the nodes at the beginning of the arcs or arrows representing the more basic or "roots" of the mishap while the nodes at the outermost limits (or the "leaves") represent the ends of the chains of events. Thus we may say that a unique chain of events or a scenario is defined by one of these outermost nodes. In our semantic network, evidence is a unit of supporting information linked by two causes. We contend that a cause is the result of another cause and this relationship is supported by evidence. In deliberation, the AMB suggests causes until a reasonable cause cannot be found. The set of resulting, connected causes is a chain of events. We also refer to a unique chain of events as a scenario.

In constructing the network from the perspective of the AMB, we recognize that the model can be used to apply "abductive" logic as well as deductive logic. Abduction allows the board to propose a scenario without evidence, effectively leaving these nodes blank. If the board uses deductive reasoning, it proposes a cause factor based on the existence of an evidence element, in effect the network which applies deductive logic

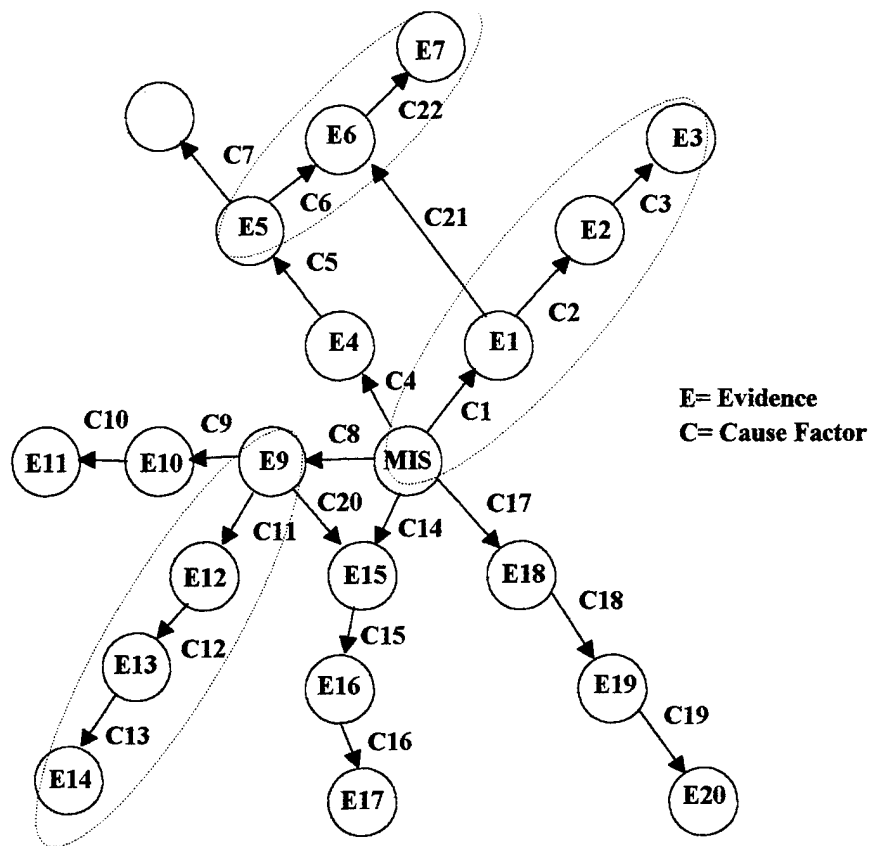


Figure 4-3 Mishap Deliberation Model

contains nodes which contain evidence. As with the strategy described by Hazard, the model should lead the AMB to ask the question "why?" in either case, testing various scenarios and then looking for evidence to support them if the evidence does not exist. If the board decides to proceed with a given chain of events such as the one represented in Figure 4-2, then they effectively populate the nodes with supporting evidence; any node that cannot be filled brings the entire chain of events into question.

Figure 4-3 is the full conceptual development of our semantic network model. This graphical representation is the development of the cognitive map taught at the Aviation Safety School, here we present it as a semantic model which we can more easily

formalize. Figure 4-3 is merely numerous branches or scenarios joined together to represent the entire problem space, or the entire mishap. As discussed in figure 4-2, each node (labeled "Ex") represents a piece of evidence and each arc (labeled "Cx") represents a cause. The dotted lines circle unique scenarios, and although only three scenarios are identified, there are nine different scenarios represented here. This semantic representation is the ideal graphical presentation for an implementation. It combines a common cognitive map with the semantic network construct which is a common artificial intelligence developmental tool, and lends itself to the textual development of the model through predicate logic we discuss below.

Although the semantic network directly relates to the investigators perceived cognitive map of the mishap, it does not directly aid our goal of the automation of the deliberation process. The network allows us to represent knowledge about a mishap graphically, but in order to automate the deliberation process we need another vehicle which allows us to generate a computer based system. In procedural language coding such as Pascal or Ada, we can use a type of "psuedo code" which attempts to make the task readable as a sentence or a statement to be translated to code. In artificial intelligence, a common approach to describing statements which can be represented by networks is predicate logic [Rowe, 1988].

Predicate logic allows us to formalize the relationships in our network and describe the assertions made during a deliberation about a mishap. Predicate logic, according to Gonzalez and Dankel, is based on the premise that *sentences* express relationships between *objects*; in our case these objects are evidences and causes. The result or the application of predicate logic is the construction of "predicates" or predicate clauses which express the relationship between certain terms. The two basic predicates we develop are the "evidence" predicate and the "probable cause" predicate. These predicates define the relationships between the terms or objects in our model which we define as "cause factors" and "evidentiary exhibits". We describe these terms in detail below.

a. Evidentiary Exhibits

In the earliest stages of a mishap, the investigative team must begin with the "raw data" or the evidentiary exhibits. Each exhibit is unique and may be in the form of a report, an interview, a photograph or even an physical piece of an aircraft. This is the mishap information in its most granular form. In our network, the evidentiary exhibit is the information which is contained in a node. As we noted earlier, the model may or may not contain evidentiary exhibits, depending on the progress of the deliberation. An evidentiary exhibit is one of the items or objects which our predicate treats.

b. Cause Factors

In the act of deliberating a mishap, the AMB will suggest a set of cause factors. A cause factor may or may not be based on evidence, but it is important the model support both abductive and deductive logical processes. To construct a set of cause factors, the board deliberates, and in deliberation the board will suggest a "root" cause factor and proceed to suggest a set of cause factors representing a chain of events. In our network representation, they would be proceeding outward away from the root and toward the ultimate leaf in a scenario, the cause factors are the arcs in the network and they establish a connection to a node. We might say that a node (evidentiary exhibit) supports the preceding cause factor was caused by the following cause factor. Referring to Figure 4-2, for example, we can say that the information contained in photo 2 was caused by the cause factor failed microswitch, and this failed microswitch (supported by the evidentiary exhibit photo 2) was caused by corrosion. The logical dialog we review in Section A of this chapter is an example of the process which produces cause factors. The cause factor, like the evidentiary exhibit is the other basic element in our predicate development.

c. Probable Causes

In defining a probable cause, we are expressing a relationship between cause factors. A probable cause is not physically represented in our network, it is rather

the basic relationship between cause factors. The distinction between a cause factor and the probable cause factor is simple: The probable cause predicate:

"probable-cause(cause-factor A, cause-factor)" states that "the probable cause of cause factor A is cause factor B". Referring back to our network, we could assert an instance as *probable-cause(failed-microswitch, corrosion)*, representing the sentence "the probable cause of the failed microswitch is corrosion. A set of these probable cause relationships and/ or a set of evidence relationships (discussed in the next section) compose our network.

Our network model implies a "transitive" relationship, in other words if a probable cause relationship states that the probable cause of cause factor A is cause factor B, then we continue the deliberation by suggesting a cause factor which caused cause factor B. If we suggest that there is an additional cause factor, C which might have caused B, then we create another probable cause entity which connects B and C. Transitivity allows us to say that if A caused B and B caused C, then C is also related to A as a cause factor. This transitivity property allows us to construct a chain of events. We apply this transitivity property by imposing some limitations on the construction of our predicates. If, for example we have a chain of events or scenario consisting of cause factors A, B, C and D in a scenario, then probable cause predicates would take the following form:

*probable-cause(cause-factor A, cause-factor B),
probable-cause(cause-factor B, cause-factor C),
probable-cause(cause-factor C, cause-factor D).*

The predicates above "chain" or connect our cause factors transitively by repeating the second cause factor as the first cause factor in the next clause. Our implementation depends on this type of relationship to ensure transitivity.

d. Evidence

Evidence, like probable cause, describes a relationship. This relationship adds an evidentiary exhibit to a probable cause relationship and thus allow the AMB to

support the given probable cause relationship. If we apply similar predicate notation to this relationship we can say "*evidence(cause factor A, cause factor B, exhibit C)*". We read this as "the cause factor A was caused by cause factor B and is supported by evidentiary exhibit C". We impose transitivity on the predicates in the same manner as we do with cause factors, only we attach the evidence element to the predicate.

e. Chain of Events

A chain of events is a set of either probable cause entities or evidence entities. The chain of events is defined by the transitive property we describe above and is composed of a "root" node and a "leaf" node. In the network in figure 4-3 we have a set of cause factors, and a set of evidentiary exhibits. The nodes represent the evidentiary exhibits and the arcs or arrows represent the cause factors. Each discrete path in this model represents a chain of events.

Ultimately, the AMB must support any given scenario with evidence, and thus the evidence predicate we propose ultimately describes all scenarios in the problem space. Although Hazard's model seems to deal with those scenarios we might call true, the board may also want to support arguments that are not true in order to document reasoning. They might need to support an argument against some suggested scenarios to disprove unsupported hearsay or illustrate the completeness of their investigation. OPNAVINST 3750.6Q requires the AMB to document this logic in the final mishap report to refute arguments which might be strongly implied but not supported by evidence. Paragraph 607 of the instruction describes how the mishap board "rejects" plausible scenarios in the text of the message that the board considers "too remote in probability".

We now continue with the implementation of the model by converting our predicate statements to Prolog code. Since Prolog is based on predicate logic, this task is not difficult and as we demonstrate, our predicates become the data elements which populate a Prolog database.

2. The Deliberation Engine

We pursue the basic implementation of the model in the Prolog computer language. The "basic" implementation is the reasoning engine described by the network model in the previous section and does not include the graphical interface which we implement in another context later in this chapter. Prolog was developed in the late 1970s and stands for PROgramming in LOGic and is based on predicate logic. It provides some distinct advantages over procedural languages when implementing problems that involve reasoning. Prolog provides mechanisms which "backtrack" rather than querying databases when attempting to satisfy a rule. The alternative to Prolog queries in procedural languages such as Pascal when trying to solve problems similar to the mishap deliberation would be numerous if-then constructs or a complicated database structure of linked lists. Prolog allows developers to directly test the truth of a query based on a set of predefined rules. In addition, the structure of the database and the rules are related to predicate calculus and developers can apply formal logic to programming without having to translate logic to a particular language procedure.

A Prolog program consists of three elements according to Clocksin and Mellish:

- ◆ Facts about objects and their relationships,
- ◆ Rules about objects and their relationships,
- ◆ Questions or queries about objects and their relationships [Clocksin and Mellish, 1984].

In keeping with our original example, we begin this discussion by presenting the Prolog database of the example presented in chapter 2, or the "facts" as described by Clocksin and Mellish. In our model, the relationship predicates we presented earlier define the facts that will populate the prolog database. The Prolog computer language is based on predicate statements similar to the examples we presented. Appendix C provides the database populated with information from our example. Two typical examples of this code are as follows:

```
prob_cause(aircraft_mishap,gear_not_down).  
evidence(aircraft_mishap,gear_not_down,wreckage_inspection).
```

The reader will recognize the relationship between prolog code and predicate logic: The code above is a reproduction of the predicates we developed applied to Prolog. This example and the complete listing in Appendix C are examples of a list of the predicate relationships. In Prolog, the database is composed of the set of assertions about the data, our database is composed of evidence and prob_cause predicates. The user of a common Prolog interpreter such as the Quintus system can either enter each of the individual predicates into an interpreter, or the user can create a file containing the list of predicates and have these facts loaded in during the interpreter initialization. Once the facts are in the Prolog database, we can query in a number of ways. Prolog allows us to simply request the contents of the database, or we can develop a set of rules which define relationships between predicates. We reproduce our model by writing a set of Prolog rules presented in Appendix C, Section B. These rules allow the user to query the database for sets of predicates describing scenarios and related evidence. Again, the user can either enter these rules into the terminal or use the Prolog "consult" routine to automatically load the rules into the system upon initialization. The code includes comments which explain much of the syntax. The reader will find consistency between the Prolog rules in the Appendix and the development of the model in the previous section. It is not the purpose of this chapter to provide an instruction in Prolog syntax, thus we provide basic descriptions of the code in the comments of the program for easier reference. The functionality of the "rules" of this program provide the user with a means to:

1. Query the contents of a scenario chain.
2. Query the final link in a scenario chain and thus determine the root cause.
3. Query any fact in the database and examine the chain which proceeds from it.
4. Retrieve an explanation for a completed and "supported" chain of events, including the evidence associated with probable causes.

Appendix C, Section C provides a scripted query result and illustrates how a set of rules we define in Prolog function once loaded into the Prolog interpreter. The first query, "prob_cause(X,Y)" requests the contents of the "prob_cause" database. As Appendix C indicates, the query simply returns a listing of prob_cause predicates in the database.

The type of query initiated above could also ask for the second half of a pair of causes. For example, if we wanted to know what cause was paired with "improper hanging" we would simply enter "prob_cause(improper_hanging, Y)" and the database would return "poor_supervision". The next query utilizes a "scenario rule" defined by the code in Appendix C, Section B. This rule is implemented by three Prolog statements, "scenario", "scenario_chain" and "scenario_chain_it". These three statements allow the user to query a list of cause_factors which compose a scenario chain. Section C of Appendix C contains a run of this type of query by requesting all scenarios in the database calling the procedures with the statement "scenario_chain(X,Y)". The script thus contains all of the scenarios in the database.

Each one of the "X" values in the above query holds the value of a cause factor, and the "Y" values contain the list of the causes which proceed from that cause factor. The reader should note that these queries are not based on evidence, thus the Prolog rules implemented here satisfy the abductive reasoning requirement mentioned earlier by allowing the user to logically "test" scenarios without providing supporting evidence. The script below continues the example run by illustrating how the user can request the cause factors emanating from a particular cause factor which begins the "chain". By entering "scenario_chain(poor_training,Y).", Prolog returns the list containing the chain or cause factors particular to "poor_training". This type of query is reproduced again with the second query.

```
scenario_chain(poor_training,Y).  
Y = [crew_uncoordinated,no_coord_training,no_command_support] ;  
No  
scenario_chain(handle_malfunctioned,Y).
```

```
Y = [improper_maintenance,omitted_step,poorly_written_handbook,  
missing_page] ;
```

We are able to ask the database the contents of a scenario relating to any cause factor. If a particular cause factor becomes the cause of more than one scenario, then the Prolog interpreter would simply list these additional scenarios. In our example above, however, both "poor_training" and "handle_malfunctioned" result in only one chain of events each.

The final Prolog Rule allows the user to query an "explanation" to a tested chain of events. The "explain_scenario" rule applies our second logical rule which attaches evidentiary exhibits to the cause_factors. In querying a root and a general cause, the user can determine if he has evidence to support the scenario. Thus the query only returns those scenarios supported by evidence. The following script illustrates a query request to support the scenario which was caused by a poorly written maintenance handbook:

```
explain_scenario(gear_not_down, poorly_written_handbook, Explain).
```

```
Explain = [gear_not_down,'was caused by',handle_malfunctioned,'and is supported  
by',wreckage_inspection,....,handle_malfunctioned,'was caused by',improper_maintenance,'and is  
supported by',records_insp,....,improper_maintenance,'was caused by',omitted_step,'and is  
supported by',handbook,....,omitted_step,'was caused by',poorly_written_handbook,'and is  
supported by',expert_interview,...] ;
```

If the scenario above contained causes that did not contain evidence, then the Prolog interpreter would return "No" indicating that the database did not satisfy the rule. The following is another example of the above type of query which examines a scenario in "mid- chain" to illustrate the a branch of a scenario which composes a separate scenario:

```
explain_scenario(no_warning_horn, corrosion, Explanation).
```

Explanation = [no_warning_horn,'was caused by',malfunctioning_switch,'and is supported by',wreckage_examination,...,malfunctioning_switch,'was caused by',corrosion,'and is supported by',wreckage_examination,...] ;

The Prolog implementation presented in this section discusses only the rules and the logical model implementation, or the reasoning "engine". The prototype implementation however, is incomplete without the addition of a user interface. Although we present an implementation of the logic in the form of a Prolog program, the user interface completes the application of logic described by our model. The interface must allow the user to communicate queries to the database and preserve the process for deliberation described by our logical model. The next section provides an example of such an interface, the reader may note that this approach and the Prolog code generated in Appendix C support an stand- alone system, but simple modifications would allow developers to adapt the same code to support the client/ server approach described in Chapter II.

3. Interface Goals, User Run-Time Routines

The importance of the user interface in the deliberation application cannot be overlooked. Even if the logic we present for the model is sound and useful for implementation, the application will be useless if the user interface is not carefully designed. When we describe the interface, we are referring to the interaction between the user and the Prolog rules presented in the previous section. The method for composing the predicate relationships that compose the "facts" in the database should be designed into the user interface so the user understands the reasoning process.

The "strategy" we mention as a goal of the deliberation application in the opening paragraphs of this chapter should be implemented as an interactive element of the user interface. The core of the application, the deliberation model written in Prolog, should be supported by a user- computer interaction which provides correctly formatted data to the Prolog interpreter or engine. This interaction should make clear the rules used in the model and present a graphical interpretation similar to the semantic model on which our

Prolog model is based. A successful interface should not only allow the user access to the database, but it should also communicate the underlying logic of the model to facilitate accurate reasoning. One of the goals of the interface should be to ensure the user has access to the decision making process as Rook and Donnell suggest. This type of interaction will extract the greatest user performance levels which in our context is accurate and timely mishap investigation [Rook and Donnell, 1993]. In addition, the interface should organize the mental model or cognitive map of the user and enable the AMB to revisit and reconstruct the cognitive maps of previous deliberations when the investigation occurs over a period of time. This recording of the deliberation process is an essential functionality: Without it the AMB might re-deliberate the mishap problem each time it reconvenes, wasting time and introducing inaccuracies. Finally, our interface should provide easy access to a database server and other applications of the system and enable the user to produce the output, the final mishap message.

4. Screen Prototype Presentation

Version 1.0B of the "Mishap Expert" is a prototype used to demonstrate the functionality of the program and the desired user interface and routines. Version 1.0B is not a client-server version, and the database is maintained entirely by the Prolog interpreter. The Prolog interpreter is a basic Edinburgh syntax interpreter which has limited windows API (Application Program Interface) capabilities; serious development of this application will require a full API enabled interpreter [See SWI reference for interpreter information]. The interface is written in Borland's Delphi, and the application itself is non-proprietary and not copyrighted. At the writing of this thesis, a team of Naval Postgraduate School students is developing a client/server version (version 1.1B) of the entire system which will include import and export of message information to and from this application. The purpose here is simply to present a basic functional interface for the Prolog implementation.

a. Mishap Expert, version 1.0B

Version 1.0B is a "bare-bones" prototype of the deliberation model presented earlier in this chapter. The purpose of presenting this basic interface is to specify and demonstrate the advantages of an effective user interface for the Prolog reasoning engine. Thus we only demonstrate basic capability in this version enabling the user to perform the functions performed by the above Prolog code. The prototype is written in Borland's Delphi using the Object Pascal language. The designed format

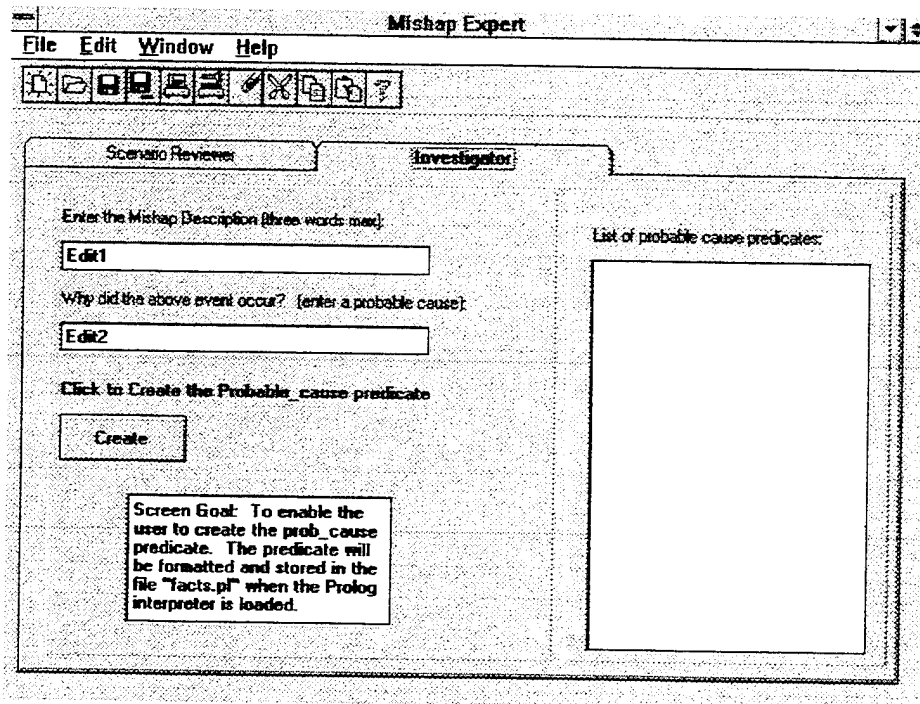


Figure 4-4 Version 1.0B Investigation Screen

includes a "Tabbed Notebook" format similar to the "Notebook" application designed by LT Hugh Brian. By clicking on any of the "Tabs" located at the top of the page, the user may navigate through the application. The screen in figure 4-4 is the Investigation screen. The primary purpose of this screen is to build the "prob_cause" predicate database for the Prolog interpreter. This screen is also designed to introduce the user to the logic of the model and serve as a tutor in the investigative logic it applies. The run-time routine we apply here is important to both the formatting of the input into Prolog code and to "educating" the user about the reasoning of the system.

First the user is prompted to enter the most general cause of the mishap, after the initial entry of the first "cause factor" the caption over the top entry box changes from "Enter the Mishap Description" to "Continue by selecting an event, or a new cause factor". In addition, a button enabling the user to create the predicate appears. When the user enters information in the top box, two events occur. First, the top box is disabled so that the user is forced into an entry into the second box. Second, the cursor moves to the second box where the user is prompted for the reason for the occurrence of the event appearing in the upper box. In this manner, the application prompts the user for the next link in the chain of events. Once the user clicks on the "create" button, the predicate is added to the list and the application makes another option available in the form of an "End Scenario" button.

As the user exits the lower entry box, the two statements are formatted and converted to Prolog syntax forming the `prob_cause` predicate which is appended onto the end of a list and eventually saved to a text file to be loaded into the interpreter in another application function. In this version of the application, the formatted statements appear in the list box to the right of the edit boxes. For example, if the user entered the system for the first time and entered the root cause as "Gear Not Down" and either exited the box or clicked on "OK" the cursor would move to the second box. When the user enters the second box, the contents of the first box dim, and that box becomes read-only. When the user enters another cause in the lower box and exits, the Prolog predicate statement appears in the list box to the right. If the user entered "Malfunctioning Handle" in the lower box and exited the box, the system would format the pair of causes and transfer the contents of the lower box to the upper box prompting the user to enter another cause. This function logically steps the user through the scenario, and also ensures the predicate arguments are identical in syntax. This matching syntax is necessary for the model "rules" to function properly. For example, the deliberation engine will relate the predicates

```
prob_cause (gear_not_down, malfunctioning_handle) ,
```


prob_cause (malfunctioning_handle, failed_microswitch

together only when it recognizes the common "malfunctioning_handle" argument. If the application does not force this type of formatting, then the model will treat each differing entry as a separate cause factor and unrelated causal pair.

Version 1.0B is a single session version with no database capabilities. Thus when the user selects the "End Scenario" button, the upper box is enabled and the user is able to begin another scenario and add to the predicate list. Version 1.1B, however uses a database and the user is forced into selecting an existing cause factor during the deliberation of a single mishap, and prohibits the user from entering a new cause factor in the top entry box. If we do not impose this limitation on the user, then the addition of a new cause factor in the top box would cause the Prolog interpreter to treat it as a separate mishap, because it would not be related to any other existing predicate in the database.

Clicking on the "Evidence Log" tab moves the user to the evidence log screen presented in figure 4-4. The evidence log simply allows the user to add, modify or delete information from a temporary file holding evidence information. The evidence log provides a repository for evidence if the user chooses not to begin deliberation immediately, this function is purely a database of evidence information. Version 1.0B reproduces the list of evidence in the Scenario Reviewer screen to build the evidence predicate saved in the "facts.pl" file. Evidence is important to the application when the user is interested in viewing the textual and graphical results of a deliberation, and figure 4-5 presents the "Scenario Reviewer" screen which constructs the Prolog evidence predicates.

The Scenario Reviewer presents the user with the list of predicates built in the Investigation screen (in the left box) and the list of evidence articles entered in the Evidence screen. By selecting prob_cause predicate and a piece of evidence and clicking "Create", the user supports a clause "cause_factor2 occurred as a result of cause_factor1" in the form of a Prolog predicate with a piece of evidence and creates the evidence predicate. If for example, the left box included a list of prob_cause statements including

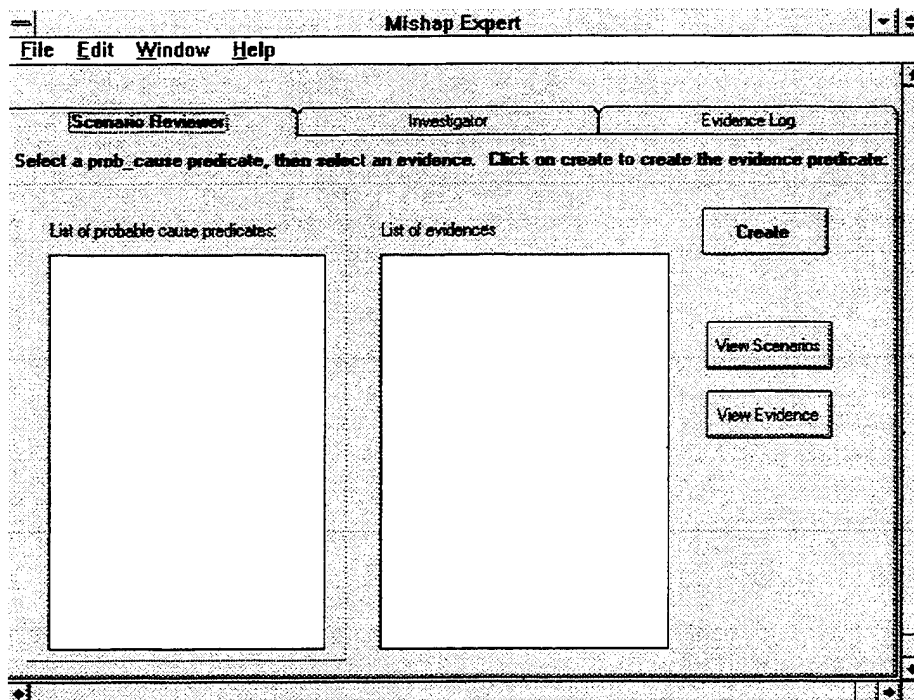


Figure 4-5 Version 1.0B scenario Reviewer Screen

the predicate `prob_cause(gear_not_down, malfunctioning handle)` and the evidence "photo 1" then the user could select both of these items and then click "Create" and the system would build the predicate `evidence(gear_not_down, malfunctioning_handle, photo1)`, and add it to the "Evid.pl" text file which stores the predicates to load into the Prolog interpreter. This functionality allows the user to construct the predicates necessary to perform the queries in a subsequent screen. By selecting the View Evidence button, he is taken to a screen which lists the contents of this text file.

When the user selects the "View Scenarios" button, he is moved to a separate application screen not in the tabbed notebook. Including the view function in the tabbed notebook might tempt the user to view the results of a query before the file has been built. The View Scenarios screen in Figure 4-6 is the query generator for the Prolog engine. The view button opens the view screen, and opens and initializes the Prolog interpreter. By "initializing" we mean the loading of the existing "factors.pl" and "evid.pl"

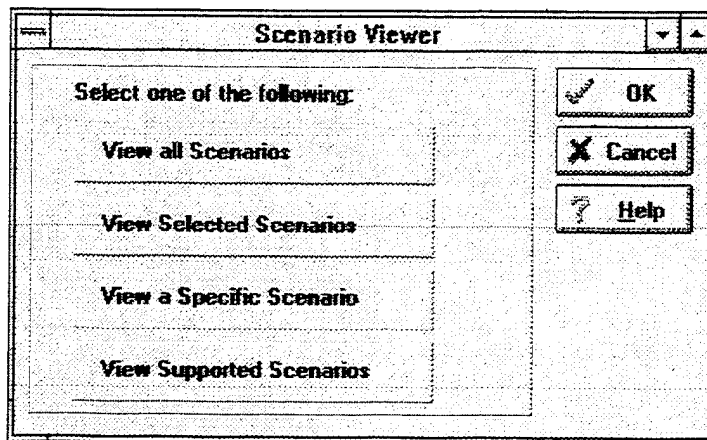


Figure 4-6 Version 1.0B Scenario Viewer Screen

files into the interpreter along with the "rules.pl" file which holds the prolog rules supporting our model. Ideally, the interpreter would be opened at this time and the application and the interpreter would communicate with API calls. Due to the limitations of our interpreter, however, we must rely on separate initializations for each button query function, returning the contents of "scripted" files recorded by the common Prolog predicate "tell". This predicate writes the results of a query to a text file, which the interface displays back to the user. In combination with the use of a custom query routine, we can attach any of the queries mentioned in the previous section to a separate button which initializes the interpreter, loads the database, performs a query and returns a value to a text file and then closes the interpreter. Appendix C, Section D provides an example of these routines along with the initialization file. The code in Appendix C is re-applied to each query generation button in the application, customized to the particular query involved. This is clearly an inefficient method only implemented here to demonstrate functionality; a better solution would utilize a Prolog interpreter which supports direct communication between applications such as OLE (Object Linking and Embedding) or DDE (Dynamic Data Exchange). This type of solution would increase the speed and efficiency of the system above the prototype implementation.

The view screen in Figure 4-6 contains four choices. The user may view all scenarios, in which the interpreter executes the `scenario(X,Y)` query and the system

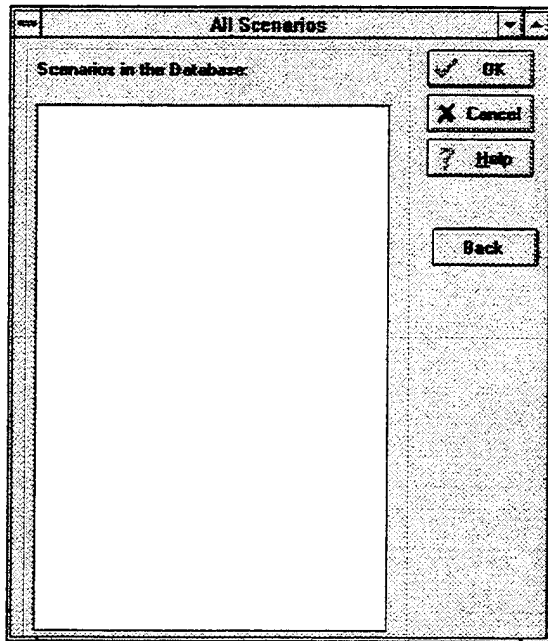


Figure 4-7 Version 1.0B All Scenario Viewer Screen

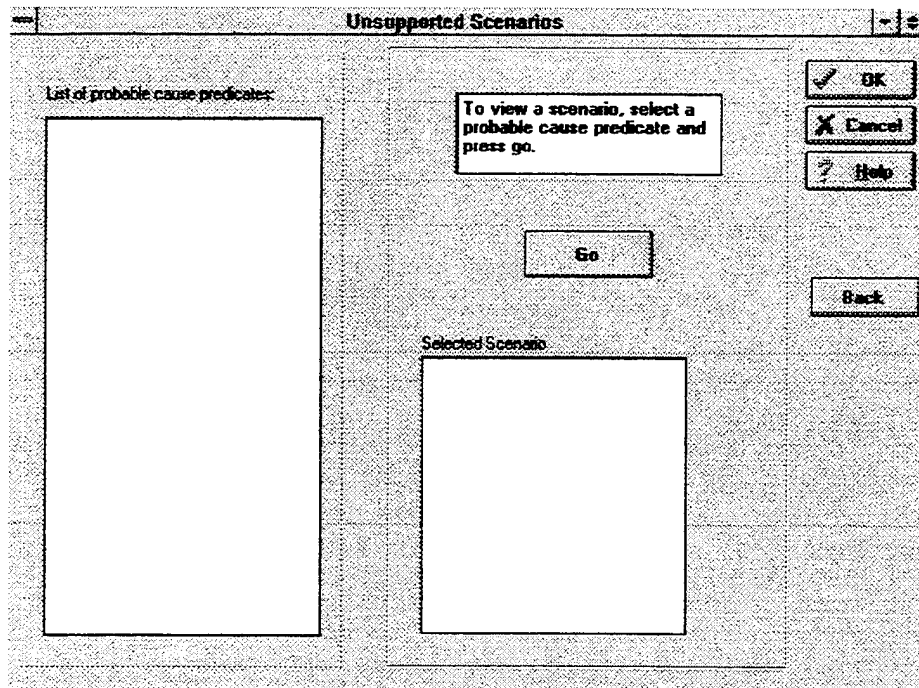


Figure 4-8 Version 1.0B Unsupported Scenario Screen

returns a list of all the developed scenarios in the box in Figure 4-7. If the user desires to view a scenario beginning with a particular event, he moves to the "unsupported scenario" screen, Figure 4-8, and selects a particular prob_cause pair. The "go" button will execute the query instantiating the X variable to the first argument of the selected prob_cause predicate. For example if the user wanted to query the prolog database about the "handle_malfunctioned, improper_maintenance" scenario as in our earlier example, he would select that predicate from the left box and click on "go". The interpreter would then execute a scenario(handle_malfunctioned, Y) query and return the same list we presented in the earlier section.

The explain scenario button executes and displays the results of the explain_scenario query described earlier in this chapter in Figure 4-9. The application presents the user with a list of existing scenario causes (the results of the scenario(X,Y) query) and the user selects one of the presented scenarios. The application takes the first and the last arguments and forms the explain_scenario(X,Y,Z) query. For example, if the

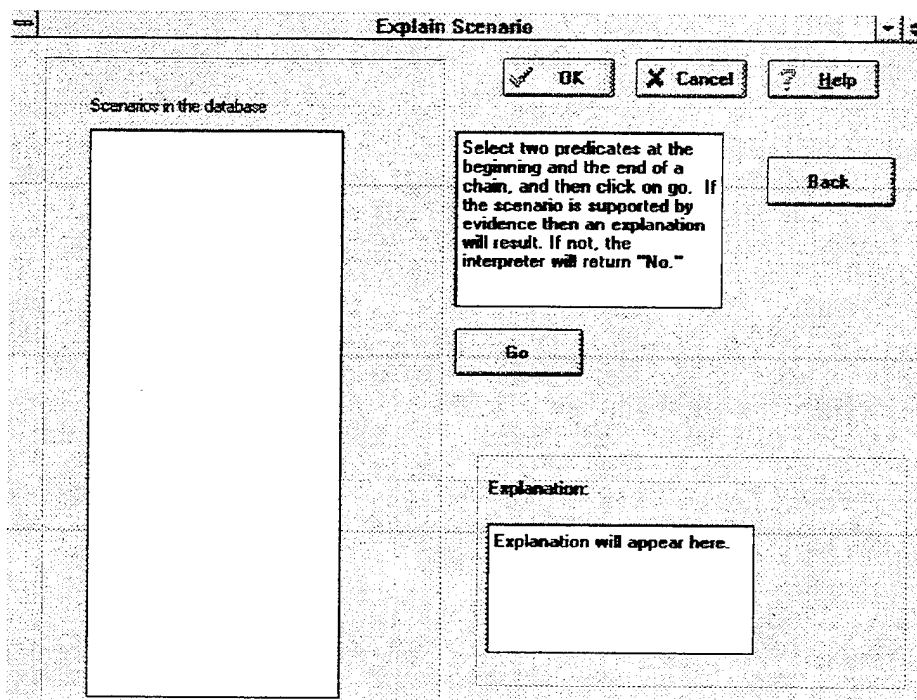


Figure 4-9 Version 1.0B Explain Scenario Screen

user selected the displayed scenario containing gear_not_down and poorly_written_handbook, the text box would display the same output generated in response to this query presented in the previous section.

b. Version 1.1B

Version 1.1B (currently under development) implements the above application with a Database Management System provided by the Borland Delphi package. The local database will enable enhanced functionality in the application and will allow the system to export information to the server in the architecture described in the Chapter III. Version 1.1B will be included as a prototype application in a project delivered to NPS faculty in partial fulfillment of the requirements for the IS 4925 course at

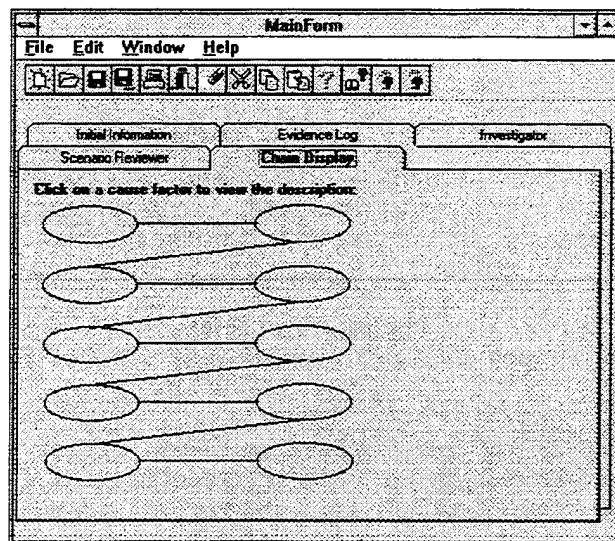


Figure 4-10 Version 1.1B Graphical Presentation Prototype

the Naval Postgraduate School.

Version 1.1B will implement the graphical description of the model important to the user interface. Figure 4-10 is a prototype of this graphical description of a chain of events and comes close to the semantic model description of our deliberation

model. This representation allows the user to view the mishap scenario in one of two contexts. The model can be called after the execution of a scenario(X,Y) query, or it can contain the contents of an explain_scenario(X,Y,Z) query. Since version 1.1B will contain a local database for query and data storage, we will be able to place the contents of related tables in the fields of the graphical description, and perform the Prolog queries on database pointers rather than formatted textual descriptions. When the user desires to view a scenario in this screen, he will see the probable causes (as he entered them) displayed above the arcs in the diagram, while the nodes will simply display the word "evidence". When the user views the same screen returning the results of an "explain_scenairio" type query, the application will label the arcs with the causes and fill in the nodes with a single word description of the evidence explaining the adjacent cause factors. This evidence text will be colored and will serve as a hypertext link to the field in the database containing the detailed description of that particular piece of evidence.

In addition to the graphical display capability described by Figure 4-10, version 1.1B will also contain the necessary "housekeeping" capabilities enabling the user to maintain the database containing the mishap information. The Prolog deliberation engine code will not require revision in files other than the initialization routines included with specific queries. The use of the database will bring the necessary closure to the entire application by allowing the user to export the results of the AMB deliberation.

When the user performs the explain_scenario query, the application will only return complete chains of events which are supported by evidence. As mentioned in Chapter II, the OPNAV 3750.6 instruction requires the AMB to both accept and reject scenarios based on supporting evidence. Thus in this context, evidence can serve to support or disprove a chain of events. In either case, the chain of events discussed in the message must be supported by evidence in order for it to appear in the message as an accepted or rejected cause factor. The MIR generation module appends the deliberation information to the initial mishap report in a formalized manner. The Mishap Expert application should contain export the evidence descriptions entered in the original

evidence log, and the data returned by the "explain_scenario" queries. The MIR generation module should allow the user to choose the queries, accept or reject them based on the content of the evidence and proceed with the remainder of the final mishap message.

V. CONCLUSION

This thesis outlined the development and implementation of a decision support system for Naval Aviation Mishap investigation and reporting, and presented the "deliberation model" for mishap investigation. In outlining the implementation, we concentrated on the model for deliberation using *artificial intelligence* methods and the Prolog language. Finally, we outlined the implementation of the model in the "Mishap Expert" application. At the writing of this thesis, the development of version 1.1b of the entire support system was ongoing, and completion of a fully functional prototype is expected by October of 1995. Version 1.0B is a basic implementation of the Mishap Expert application only. This application prototype presents the Prolog model with a very basic user interface which demonstrates desired routines. A full implementation of this system requires significant further study and many of these important issues are beyond the scope of this thesis.

A. IMPLEMENTATION ISSUES

This thesis presents a largely experimental implementation of the DSS and the deliberation application. The full implementation of a similar system for fleet use requires careful study and consideration of such issues as infrastructure supporting a client/ server architecture and compatibility with different platforms. In addition, the selection of the database system and the related applications (such as the Prolog interpreter) require careful attention. An informal survey of aviation squadrons indicates that this infrastructure does not exist in most cases, and thus the client/ server approach is presently infeasible. We suggest that the most practical application of this model is therefore a single-platform system supported by a database as described in Chapter III.

The use of Prolog and Borland's Delphi in this prototype should not suggest that this is the only implementation approach. Developers may find solutions using other languages and tools. The commercial Prolog market continues to grow and at this writing, powerful and complete graphical Prolog development tools are appearing,

hopefully future development of this model will result in an single application providing reasoning and a user interface rather than the use of multiple applications demonstrated in our prototype.

B. QUESTIONS FOR FURTHER STUDY

The obvious subject for further study in this thesis is the full implementation and testing of the entire Decision Support System. As mentioned in Chapter IV, this research is ongoing and will be completed by October of 1995. In addition to the implementation of the system, we suggest further questions for study:

- ◆ The implementation and testing of the entire decision support system.
- ◆ An examination of the computing infrastructure necessary to support such a system.
- ◆ How would the department of the Navy support this software implementation?

This question will become increasingly important as end- users continue to develop useful information applications. If the support for end- user applications is not specified, users might become dependent on an application with no recourse when the application or system fails or requires update. In applications where processes are automated such as the message generation capabilities of our system, the absence of an automated system might paralyze an organization that forgets how to manually execute automated processes. This question is particularly important to DoD when military officers develop applications and subsequently move to other assignments, taking the "support" for an application with them.

- ◆ Additional analysis of the deliberation model, with continued discussions with the Aviation Safety School. This model may also be appropriate for entry into the OPNAVINST 3750.6Q instruction as a method for investigation.

As we discussed in the introduction of this thesis, we presented the development of the administrative system architecture to the end-user developer recognizing the growing trend of end-user application development with such tools as Borland's Delphi. As tools become available to end-users to develop applications with increasing ease, examinations of processes and practices such as the mishap deliberation process we present here will

become more important. The "addiction" to the power of Rapid Application Development (RAD) tools will lead to an explosion of applications which do not support or improve a defined process. Thus process examination in application development will become more important than application implementation.

Our discussion directly addressed a weakness in the Naval Aviation Safety mishap investigation process. Although the Aviation Safety School at the Naval Postgraduate School informally teaches a deliberation methodology, this methodology is not formalized in the directive instruction, OPNAVINST 3750.6Q. In suggesting a strategy based on the teaching methods at the NPS Safety school, we provide an avenue for improved mishap investigation and reporting through the use of an automated "deliberation" model. In addition, we present the architecture for an administrative system to support the entire reporting process.

As the Navy continues to downsize and conserve resources, the preservation of both human and material assets in naval aviation will receive greater attention. The thoughtful examination of the investigation process and the application of appropriate technology can enhance and improve mishap investigation and reporting. By supporting the administrative demands of an investigation and providing a defined process for deliberation, an implementation of the model discussed here will not only improve the efficiency of the investigation process, it will also improve the quality of the completed investigation.

In conclusion, mishaps are an unfortunate yet inevitable product of the practice of aviation. Naval aviation in particular provides the most challenging and demanding environment in the world of aviation. From the flight decks of carriers to the cockpits of patrol aircraft far from base, naval aviators perform feats which put them at far greater risk than other aviation communities, and thus the task of mishap investigation in naval aviation is far more difficult and critical to their survival. The occurrence of a mishap obligates us to rectify undetected hazards through investigation, failure to detect these hazards dooms us to eventually repeating our mistakes and suffering the consequences

over and over. Each mishap is thus an opportunity to save future lives and assets, but if we fail to logically examine evidence or consider all possible scenarios in a mishap, we endanger the future of Naval Aviation.

LIST OF REFERENCES

- Axelrod, R. *Structure of Decision*, Princeton University Press, Princeton, New Jersey, 1976.
- Clocksin, William F. and Christopher S. Mellish, *Programming in Prolog*, Springer-Verlag, Berlin Heidelberg, Germany, 1984.
- Duncan, Nancy and David Paradice, "Creativity in GDSS: An Integration of Creativity, Group Dynamics, and Problem Solving Theory." Proceedings of the 25th Hawaii International Conference on Systems Sciences pp. 277-287, Vol. 4, IEEE Computer Society Press, Los Alamitos, CA, 1991.
- Eden, Colin, "Strategic Decision Support Through Computer Based Analysis and Presentation of Cognitive Maps." Working Paper, Management, Science, Theory Method and Practice Series, September, 1988.
- Frew, Barry, Instructional Notes for IS4182, Naval Postgraduate School, 1995.
- Gonzales, Avelino and Douglas D. Dankel, *The Engineering of Knowledge- Based Systems, Theory and Practice*, Prentice- Hall, Englewood Cliffs, New Jersey, 1993.
- Harkey, Dan, Robert Orfali and Jeri Edwards, *Essential Client/ Server Survival Guide*, John Wiley & Sons, New York, 1994.
- Quillan, M. R., "Semantic Memory." In *Semantic Information Processing*, ed. M. Minsky. Cambridge, MA: MIT Press, 1968.
- Reiger, C., and M. Grinberg, "The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms." Proceedings of the Fifth International Joint Conference on Artificial Intelligence. San Mateo, CA: Morgan Kaufmann, 1977.
- Rook, Frederick W. and Michael L. Donnell, "Human Cognition and the Expert System Interface: Mental Models and Inference Explanations", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 23 No. 6, November/ December 1993.
- Rowe, N.C., *Artificial Intelligence Through Prolog*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
- Weilemaker, Jan, SWI-PROLOG Version 1.9.4, University of Amsterdam, Amsterdam, The Netherlands, 1994. Available on the internet as freeware at <http://www.psy.uva.nl>.

Zhang, When-Ran, "Cognitive Map Composition, Derivation, and Focus Generation for Distributed Group Decision Support", Proceedings of the 23rd Annual Hawaii International Conference on Systems Sciences, pp. 318-326, vol.3. IEEE Computer Society Press, Los Alamitos, CA USA, 1994.

APPENDIX A

DATABASE DESIGN

A. ANSI SQL-92 SCHEMA

CREATE SCHEMA MisDataBase

CREATE TABLE Mishap

```
( Mishap_Serial_Num      SMALLINT NOT NULL,
  EventDate              DATE NOT NULL,
  EventTime              TIME NOT NULL,
  Time_Zone              CHARACTER VARYING (10) NOT NULL,
  Location               CHARACTER VARYING (10),
  Unit_ID_FK2            INTEGER NOT NULL,
  Flight_Information_ID_FK3  INTEGER,
  Mishap_Summary         CHARACTER VARYING (10) NOT NULL,
  _ID                    INTEGER NOT NULL,
```

```
PRIMARY KEY ( _ID ),
UNIQUE ( Mishap_Serial_Num ),
FOREIGN KEY ( Unit_ID_FK2 )
REFERENCES Unit,
FOREIGN KEY ( Flight_Information_ID_FK3 )
REFERENCES Flight_Information
```

)

CREATE TABLE Mishap_Recommendations

```
( Recommendations      CHARACTER VARYING (32000) NOT NULL,
  Mishap_ID_FK1         INTEGER NOT NULL,
  _ID                    INTEGER NOT NULL,
```

```
PRIMARY KEY ( _ID ),
FOREIGN KEY ( Mishap_ID_FK1 )
REFERENCES Mishap
```

)

CREATE TABLE Passenger

```
( Person_ID_FK11        INTEGER NOT NULL,
  Rank                  CHARACTER VARYING (10),
  Rate                  CHARACTER VARYING (10),
  Service               CHARACTER VARYING (10),
  USN                   CHARACTER VARYING (10),
  Non_DoD               CHARACTER VARYING (10),
  Duty_Status           CHARACTER VARYING (10),
  Parent_Unit           CHARACTER VARYING (10),
  Lost_Wk_Days          SMALLINT,
  Hospital_Days         SMALLINT,
  Mishap_ID_FK14        INTEGER,
```



```

_ID          INTEGER NOT NULL,

PRIMARY KEY (_ID),
FOREIGN KEY ( Person_ID_FK11 )
REFERENCES Person,
FOREIGN KEY ( Mishap_ID_FK14 )
REFERENCES Mishap
)

CREATE TABLE Unit
( Short_Name          CHARACTER VARYING (10) NOT NULL,
  UIC                 CHARACTER VARYING (10) NOT NULL,
  PLAD                CHARACTER VARYING (10),
  Address_1           CHARACTER VARYING (10) NOT NULL,
  Mishap_Number       CHARACTER VARYING (10) NOT NULL,
  _ID                 INTEGER NOT NULL,

PRIMARY KEY (_ID),
UNIQUE ( UIC )
)

CREATE TABLE Flight_Information
( Origin              CHARACTER VARYING (10),
  Flight_Pur_Code     CHARACTER VARYING (10) NOT NULL,
  Mission             CHARACTER VARYING (10) NOT NULL,
  Destination         CHARACTER VARYING (10) NOT NULL,
  AirCraft_Evolution CHARACTER VARYING (10) NOT NULL,
  Type_Flight_Plan   CHARACTER VARYING (3) NOT NULL,
  Day_Night           CHARACTER VARYING (5) NOT NULL,
  WX_Desc             CHARACTER VARYING (32000) NOT NULL,
  Altitude            SMALLINT NOT NULL,
  _ID                 INTEGER NOT NULL,

PRIMARY KEY (_ID)
)

CREATE TABLE Mishap_Civilian_Fatalities
( Civilian_Fatalities CHARACTER VARYING (10) NOT NULL,
  Mishap_ID_FK4        INTEGER NOT NULL,
  _ID                  INTEGER NOT NULL,

PRIMARY KEY (_ID),
FOREIGN KEY ( Mishap_ID_FK4 )
REFERENCES Mishap
)

CREATE TABLE Mishap_DoD_Fatalities
( DoD_Fatalities      CHARACTER VARYING (10) NOT NULL,
  Mishap_ID_FK5       INTEGER NOT NULL,
  _ID                 INTEGER NOT NULL,

PRIMARY KEY (_ID),

```

```

FOREIGN KEY ( Mishap_ID_FK5 )
REFERENCES Mishap
)

CREATE TABLE Crewmember
( Person_ID_FK7          INTEGER NOT NULL,
Rank                   CHARACTER VARYING (10),
Rate                   CHARACTER VARYING (10),
Service                CHARACTER VARYING (10) NOT NULL,
Parent_Unit            CHARACTER VARYING (10) NOT NULL,
Duty_Status            CHARACTER VARYING (10) NOT NULL,
Hospital_Days          SMALLINT,
NVG_Use                CHARACTER VARYING (10) NOT NULL,
Model_Hours            INTEGER,
Total_Hours            INTEGER,
Mishap_ID_FK10         INTEGER NOT NULL,
_ID                    INTEGER NOT NULL,

PRIMARY KEY ( _ID ),
FOREIGN KEY ( Person_ID_FK7 )
REFERENCES Person,
FOREIGN KEY ( Mishap_ID_FK10 )
REFERENCES Mishap
)

CREATE TABLE Evidence
( Enclosure_Number     CHARACTER VARYING (4) NOT NULL,
Description             CHARACTER VARYING (10),
Summary_Number         BIT (8) NOT NULL,
Summary                CHARACTER VARYING (32000),
Mishap_ID_FK16         INTEGER,
_ID                    INTEGER NOT NULL,

PRIMARY KEY ( _ID ),
UNIQUE ( Enclosure_Number ),
FOREIGN KEY ( Mishap_ID_FK16 )
REFERENCES Mishap
)

CREATE TABLE Factor
( Factor_Name           CHARACTER VARYING (10) NOT NULL,
Accept_or_Reject       CHARACTER VARYING (10) NOT NULL,
Explanation            CHARACTER VARYING (10) NOT NULL,
Mishap_ID_FK17         INTEGER,
_ID                    INTEGER NOT NULL,

PRIMARY KEY ( _ID ),
UNIQUE ( Factor_Name ),
FOREIGN KEY ( Mishap_ID_FK17 )
REFERENCES Mishap
)

```

```

CREATE TABLE Causal_Factor
( Determination_Statement CHARACTER VARYING (10) NOT NULL,
  Factor_ID_FK18           INTEGER NOT NULL,
  RAC                      CHARACTER VARYING (3) NOT NULL,
  Who_or_Comp              CHARACTER VARYING (10) NOT NULL,
  What_or_Mode             CHARACTER VARYING (10) NOT NULL,
  Why_or_Agent             CHARACTER VARYING (10) NOT NULL,
  Para_Number              CHARACTER VARYING (10) NOT NULL,
  Mishap_ID_FK19           INTEGER,
  _ID                      INTEGER NOT NULL,

  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Factor_ID_FK18 )
    REFERENCES Factor,
  FOREIGN KEY ( Mishap_ID_FK19 )
    REFERENCES Mishap
)

```

```

CREATE TABLE CF_Other_Dam
( Determination_Statement CHARACTER VARYING (10) NOT NULL,
  Factor_ID_FK20           INTEGER NOT NULL,
  Who_Comp                 CHARACTER VARYING (10) NOT NULL,
  What_or_Mode             CHARACTER VARYING (10),
  Why_or_Agent             CHARACTER VARYING (10),
  Mishap_ID_FK21           INTEGER,
  _ID                      INTEGER NOT NULL,

  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Factor_ID_FK20 )
    REFERENCES Factor,
  FOREIGN KEY ( Mishap_ID_FK21 )
    REFERENCES Mishap
)

```

```

CREATE TABLE Person
( Name                     CHARACTER VARYING (10) NOT NULL,
  SSN                      CHARACTER VARYING (10) NOT NULL,
  Address_1                CHARACTER VARYING (10),
  Phone_2                  CHARACTER VARYING (10),
  Officer_Recall_ID_FK6    INTEGER,
  _ID                      INTEGER NOT NULL,

  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Officer_Recall_ID_FK6 )
    REFERENCES Officer_Recall
)

```

```

CREATE TABLE Officer_Recall
( Position_AA              CHARACTER VARYING (10),
  _ID                      INTEGER NOT NULL,

  PRIMARY KEY ( _ID )
)

```

```

)

CREATE TABLE Crewmember_Position
( Position_AA          CHARACTER VARYING (10) NOT NULL,
  Crewmember_ID_FK8    INTEGER NOT NULL,
  _ID                  INTEGER NOT NULL,

  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Crewmember_ID_FK8 )
    REFERENCES Crewmember
)

CREATE TABLE Crewmember_Signifigant_Injuries
( Signifigant_Injuries CHARACTER VARYING (10) NOT NULL,
  Crewmember_ID_FK9     INTEGER NOT NULL,
  _ID                  INTEGER NOT NULL,

  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Crewmember_ID_FK9 )
    REFERENCES Crewmember
)

CREATE TABLE Passenger_Position
( Position_AA          CHARACTER VARYING (10) NOT NULL,
  Passenger_ID_FK12    INTEGER NOT NULL,
  _ID                  INTEGER NOT NULL,

  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Passenger_ID_FK12 )
    REFERENCES Passenger
)

CREATE TABLE Passenger_Signifigant_Injuries
( Signifigant_Injuries CHARACTER VARYING (10) NOT NULL,
  Passenger_ID_FK13    INTEGER NOT NULL,
  _ID                  INTEGER NOT NULL,

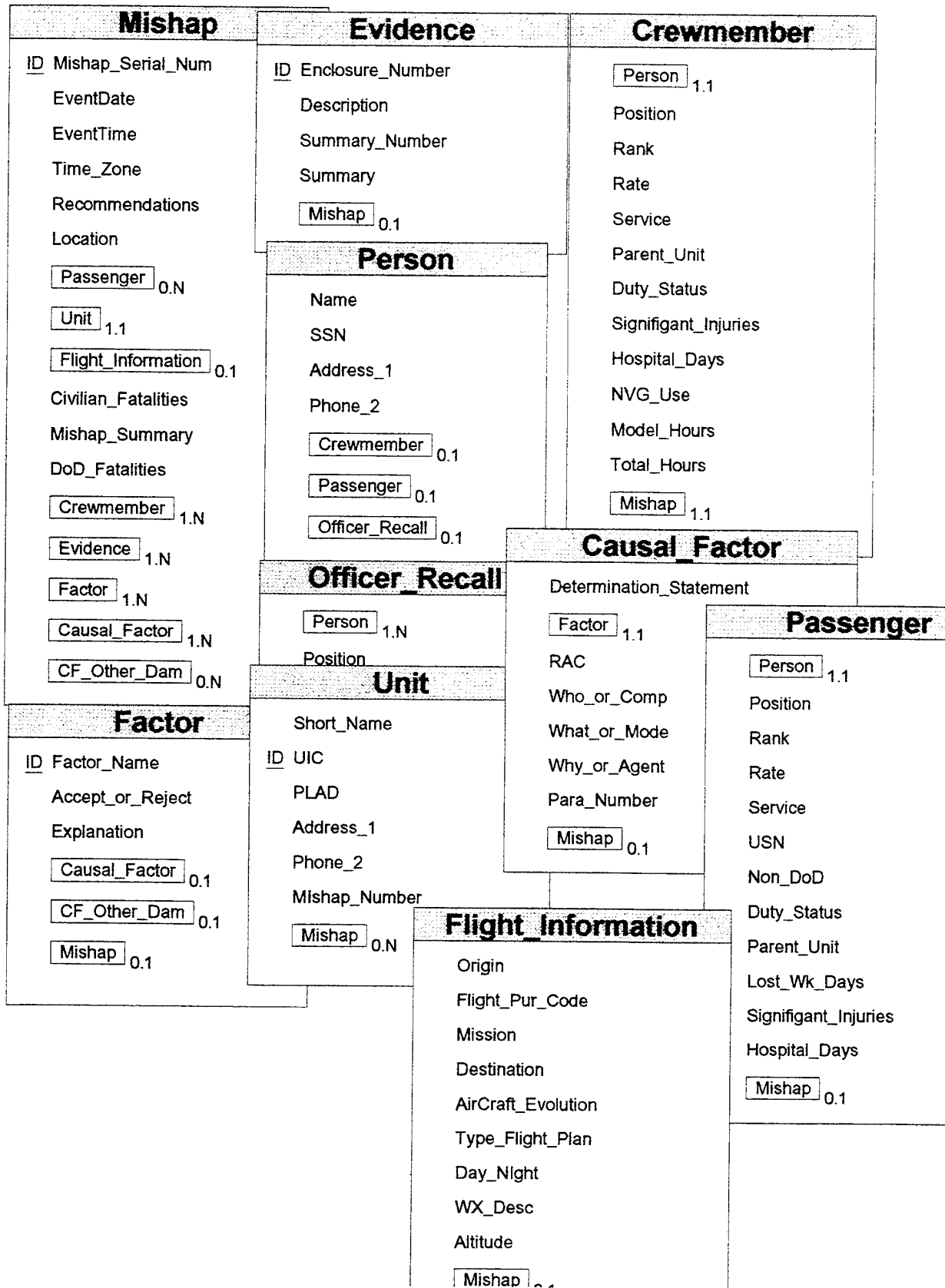
  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Passenger_ID_FK13 )
    REFERENCES Passenger
)

CREATE TABLE Unit_Phone_2
( Phone_2             CHARACTER VARYING (10) NOT NULL,
  Unit_ID_FK15        INTEGER NOT NULL,
  _ID                 INTEGER NOT NULL,

  PRIMARY KEY ( _ID ),
  FOREIGN KEY ( Unit_ID_FK15 )
    REFERENCES Unit
)

```

B. SEMANTIC OBJECT REPRESENTATION



APPENDIX B

NOTEBOOK SOURCE CODE

(WRITTEN BY LT HUGH BRIAN, USN)

```
unit Noteform;

interface

uses SysUtils, WinTypes, WinProcs, Classes, Graphics, Forms, Controls, Menus,
    Dialogs, StdCtrls, Buttons, ExtCtrls, TabNotBk, Grids, IniFiles, Tabs, Spin,
    Printers, Gauges, Aboutnot, Mask, Diag1, Printgd;

type
TNoteBookForm = class(TForm)
    MainMenu: TMainMenu;
    FileMenu: TMenuItem;
    SaveItem: TMenuItem;
    ExitItem: TMenuItem;
    N1: TMenuItem;
    SaveDialog: TSaveDialog;
    Help1: TMenuItem;
    About1: TMenuItem;
    StatusBar: TPanel;
    Notebook: TTabbedNotebook;
    Grid1: TStringGrid;
    MishapPanel: TPanel;
    MishapCategory: TComboBox;
    Yes1: TButton;
    Panel1: TPanel;
    MishapSeverity: TComboBox;
    SeverityDone: TButton;
    OprepMemo: TMemo;
    GroupBox7: TGroupBox;
    OprepPOC: TComboBox;
    OprepType: TComboBox;
    Notebook1: TNotebook;
    Label7: TLabel;
    Label8: TLabel;
```

Label10: TLabel;
CallerName: TEdit;
CallerOrganization: TEdit;
MishapLocation: TEdit;
GroupBox3: TGroupBox;
MishapDescriptionMemo: TMemo;
GroupBox11: TGroupBox;
Weather: TMemo;
DayNiteSel: TRadioGroup;
GroupBox13: TGroupBox;
Label35: TLabel;
Label36: TLabel;
Label37: TLabel;
Label38: TLabel;
Label40: TLabel;
Origin: TEdit;
MissionCode: TComboBox;
FPC: TComboBox;
Destination: TEdit;
AirEvol: TEdit;
TabSet1: TTabSet;
Panel5: TPanel;
UnitName: TLabel;
Label3: TLabel;
Address: TLabel;
Label2: TLabel;
Label15: TLabel;
AircraftTypeLabel: TLabel;
SquadronName: TEdit;
UnitPlad: TEdit;
SAddress: TEdit;
SState: TEdit;
AircraftTypeComboBox: TComboBox;
GroupBox2: TGroupBox;
CheckBox2: TCheckBox;
CheckBox3: TCheckBox;
CheckBox1: TCheckBox;
CheckBox4: TCheckBox;
CheckBox5: TCheckBox;
CheckBox6: TCheckBox;
CheckBox7: TCheckBox;
CheckBox8: TCheckBox;
CheckBox9: TCheckBox;

CheckBox11: TCheckBox;
SCVoice: TMemo;
Timer1: TTimer;
POCPhone: TComboBox;
CadPlad: TComboBox;
TYCOM: TComboBox;
WingCom: TComboBox;
Label4: TLabel;
Label9: TLabel;
DodDead: TRadioGroup;
CivDead: TRadioGroup;
Grid2: TStringGrid;
Gauge1: TGauge;
Label11: TLabel;
UpdateUnitInfo: TButton;
Label13: TLabel;
Label14: TLabel;
GroupBox5: TGroupBox;
Label5: TLabel;
Label1: TLabel;
Label6: TLabel;
LocalTime: TEdit;
ZuluTime: TEdit;
DateAndTime: TEdit;
UpdateTime: TButton;
Label17: TLabel;
ZuluTimeSet: TSpinEdit;
Label18: TLabel;
LatLong: TEdit;
Label19: TLabel;
FlightPlan: TComboBox;
Label22: TLabel;
Altitude: TEdit;
Label39: TLabel;
ACGrid: TStringGrid;
SaveOprepVoiceReport1: TMenuItem;
OPREPMessage1: TMenuItem;
SafetyCenterVoiceReport1: TMenuItem;
MishapReport1: TMenuItem;
Contents1: TMenuItem;
Panel2: TPanel;
UpdateMisAircraft: TButton;
AircraftNumber: TSpinEdit;

Label23: TLabel;
Panel3: TPanel;
Button3: TButton;
NumberCrew: TSpinEdit;
Label24: TLabel;
PersGrid: TStringGrid;
ShipPlad: TEdit;
Label12: TLabel;
ComPhoneLabel: TLabel;
AutovonLabel: TLabel;
LTime: TEdit;
ZTime: TEdit;
Print1: TMenuItem;
OprepVoiceReport1: TMenuItem;
OPREPMessage2: TMenuItem;
SafetyCenterVoiceReport2: TMenuItem;
MishapReport2: TMenuItem;
PrintDialog1: TPrintDialog;
N2: TMenuItem;
StartTimer: TButton;
Panel4: TPanel;
Button2: TButton;
PaxGrid: TStringGrid;
TabContinue: TButton;
MisSevQ1: TRadioGroup;
MisSevQ2: TRadioGroup;
MisSevQ3: TRadioGroup;
MisSevQ4: TRadioGroup;
MisSevQ5: TRadioGroup;
Damage: TRadioGroup;
Injury: TRadioGroup;
CheckBox12: TCheckBox;
MishapNumber: TSpinEdit;
Label27: TLabel;
ServiceSel: TRadioGroup;
FleetCom: TComboBox;
Label20: TLabel;
UniCom: TComboBox;
Label21: TLabel;
UIC: TEdit;
Label29: TLabel;
Label30: TLabel;
Panel7: TPanel;

Panel8: TPanel;
MsgMemo: TMemo;
OprepRemarks: TMemo;
Panel9: TPanel;
Panel10: TPanel;
Panel6: TPanel;
Label26: TLabel;
ALSSBox: TCheckBox;
CarLandBox: TCheckBox;
HeloLandBox: TCheckBox;
SarCheckBox: TCheckBox;
MishapReport: TMemo;
Panel11: TPanel;
GroupBox1: TGroupBox;
Label16: TLabel;
PaxNumber: TSpinEdit;
GroupBox4: TGroupBox;
Label31: TLabel;
InjuredPax: TSpinEdit;
GroupBox6: TGroupBox;
Label32: TLabel;
InjuredNonOccupants: TSpinEdit;
AVPhone: TMaskEdit;
ComPhone: TMaskEdit;
Szip: TEdit;
Panel13: TPanel;
OfficerRecall1: TMenuItem;
BitBtn3: TBitBtn;
BitBtn1: TBitBtn;
BitBtn2: TBitBtn;
BitBtn4: TBitBtn;
BitBtn5: TBitBtn;
BitBtn6: TBitBtn;
BitBtn7: TBitBtn;
BitBtn8: TBitBtn;
BitBtn9: TBitBtn;
BitBtn10: TBitBtn;
BitBtn11: TBitBtn;
BitBtn12: TBitBtn;
BitBtn13: TBitBtn;
BitBtn14: TBitBtn;
BitBtn15: TBitBtn;
BitBtn16: TBitBtn;

BitBtn18: TBitBtn;
BitBtn19: TBitBtn;
BitBtn20: TBitBtn;
BitBtn21: TBitBtn;
BitBtn22: TBitBtn;
Panel12: TPanel;
UpdateOfficer: TButton;
RecallMemo: TMemo;
BitBtn17: TBitBtn;
BitBtn23: TBitBtn;
BitBtn24: TBitBtn;
procedure ShowHint(Sender: TObject);
procedure ExitItemClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure UpdateOfficerClick(Sender: TObject);
procedure MishapCatDoneClick(Sender: TObject);
procedure TabSet1Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure StartTimerClick(Sender: TObject);
procedure UpdateOprepClick(Sender: TObject);
procedure OprepPOCChange(Sender: TObject);
procedure UpdateOPClick(Sender: TObject);
procedure FormResize(Sender: TObject);
procedure UpdateUnitInfoClick(Sender: TObject);
procedure UpdateSafetyCallClick(Sender: TObject);
procedure UpdateMRClick(Sender: TObject);
procedure UpdateTimeClick(Sender: TObject);
procedure UpdateMishapClick(Sender: TObject);
procedure AircraftNumberChange(Sender: TObject);
procedure NumberCrewChange(Sender: TObject);
procedure About1Click(Sender: TObject);
procedure InjuredPaxChange(Sender: TObject);
procedure UpdateMisAircraftClick(Sender: TObject);
procedure Contents1Click(Sender: TObject);
procedure MishapReport1Click(Sender: TObject);
procedure SafetyCenterVoiceReport1Click(Sender: TObject);
procedure SaveOprepVoiceReport1Click(Sender: TObject);
procedure OPREPMMessage1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure AircraftTypeComboBoxChange(Sender: TObject);
procedure OprepVoiceReport1Click(Sender: TObject);
procedure OPREPMMessage2Click(Sender: TObject);
procedure SafetyCenterVoiceReport2Click(Sender: TObject);

```

procedure MishapReport2Click(Sender: TObject);
procedure TabContinueClick(Sender: TObject);
procedure InfoCompClick(Sender: TObject);
procedure MishapInfoCompleteClick(Sender: TObject);
procedure CompeteNextClick(Sender: TObject);
procedure Button7Click(Sender: TObject);
procedure Yes1Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure GotoMishapCatClick(Sender: TObject);
procedure CompleteMishapCatClick(Sender: TObject);
procedure MishapSevDoneClick(Sender: TObject);
procedure Button8Click(Sender: TObject);
procedure MRCompleteClick(Sender: TObject);
procedure OfficerRecall1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
NoteBookForm: TNoteBookForm;

type TMishapClass = (ClassA,ClassB,ClassC);
const
ScreenWidth: LongInt = 640; {I designed my form in800x600 mode.}
ScreenHeight: LongInt = 480;

implementation

{$R *.DFM}

procedure TNoteBookForm.ShowHint(Sender: TObject);
begin
  StatusBar.Caption := Application.Hint;
end;

procedure TNoteBookForm.ExitItemClick(Sender: TObject);
begin
  Close;
end;

```

```

end;

procedure TNoteBookForm.FormCreate(Sender: TObject);
var
Intro: TIntroForm; {IntroForm pusedo splash screen}
NotebookIni: TIniFile;
R,C,I: Integer; {Counters}
Name, Phone: String;
UnitInfoList: TStringList;
x, y: LongInt; {Integers will not not a large enough value.}
begin
{Screen Settings}
{this comes to you with courtesey from Loyd of Borland Tech Support}
NoteBookForm.scaled := true;
x := getSystemMetrics(SM_CXSCREEN);
y := getSystemMetrics(SM_CYSCREEN);
if (x <> ScreenHeight) or (y <> ScreenWidth) then
begin
NoteBookForm.height := NoteBookForm.height * y DIV ScreenHeight;
NoteBookForm.width := NoteBookForm.width * x DIV ScreenWidth;
scaleBy(x, ScreenWidth);
end;
{End Screen Settings}

{Initial Settings }
TabSet1.Tabs := Notebook1.Pages;
Notebook.PageIndex:=0;
Application.OnHint := ShowHint;
Label11.Transparent:=True;
{set combo boxes to first item}
for i:= 0 to GroupBox7.ControlCount -1 do
TComboBox(GroupBox7.Controls[i]).ItemIndex:=0;
{Update time and date stamp when program loads}
UpdateTime.Click;
{Start mishap timer}
StartTimer.Click;
{Create List for retrieving valuse from Ini File}
UnitInfoList:=TStringList.Create;
{NoteBok.ini file initialization}
NoteBookIni:= TIniFile.Create('notebook.ini');
for R:= 1 to Grid1.RowCount-1 do begin
Grid1.Cells[1,R]:= NoteBookIni.ReadString('OfficerRecall', 'Name' +
IntToStr(R),'NAME');

```

```

Grid1.Cells[2,R]:= NotebookIni.ReadString('OfficerRecall', 'UnitId' +
IntToStr(R),SquadronName.Text);
Grid1.Cells[3,R]:= NotebookIni.ReadString('OfficerRecall', 'CPhone' +
IntToStr(R),'COMPHONE');
Grid1.Cells[4,R]:= NotebookIni.ReadString('OfficerRecall', 'APhone' +
IntToStr(R),'AUTOVON#');
end;
for R:= 1 to Grid1.RowCount-1 do begin
Grid2.Cells[1,R]:= NotebookIni.ReadString('MisBoard', 'Name' +
IntToStr(R),'NAME');
Grid2.Cells[2,R]:= NotebookIni.ReadString('MisBoard', 'UnitId' +
IntToStr(R),SquadronName.Text);
Grid2.Cells[3,R]:= NotebookIni.ReadString('MisBoard', 'CPhone' +
IntToStr(R),'COMPHONE');
Grid2.Cells[4,R]:= NotebookIni.ReadString('MisBoard', 'APhone' +
IntToStr(R),'AUTOVON#');
end;

```

```

{Read in UnitInfo into TStringList}

```

```

NotebookIni.ReadSectionValues('UnitInfo',UnitInfoList);

```

```

{Assign values from List to edit boxes}

```

```

SquadronName.Text:=UnitInfoList.Values['UnitName'];

```

```

UnitPlad.Text:=UnitInfoList.Values['UnitPlad'];

```

```

SAddress.Text:=UnitInfoList.Values['Address'];

```

```

SState.Text:=UnitInfoList.Values['State'];

```

```

SZip.Text:=UnitInfoList.Values['Zip'];

```

```

ShipPlad.Text:=UnitInfoList.Values['ShipPlad'];

```

```

AircraftTypeComboBox.Text:=UnitInfoList.Values['AircraftType'];

```

```

CadPlad.Text:=UnitInfoList.Values['CADPlad'];

```

```

AVPhone.Text:=UnitInfoList.Values['AVPhone'];

```

```

ComPhone.Text:=UnitInfoList.Values['ComPhone'];

```

```

TYCOM.Text:=UnitInfoList.Values['Tycom'];

```

```

Wingcom.Text:=UnitInfoList.Values['Wingcom'];

```

```

FleetCom.Text:= UnitInfoList.Values['FleetCom'];

```

```

UniCom.Text:= UnitInfoList.Values['UnifiedCom'];

```

```

ZuluTimeSet.Value:=NotebookIni.ReadInteger('UnitInfo','ZuluTimeSet', 0);

```

```

MishapNumber.Value:=NotebookIni.ReadInteger('UnitInfo','MishapNumber', 0);

```

```

ServiceSel.ItemIndex:=NotebookIni.ReadInteger('UnitInfo','ServiceSel',0);

```

```

UIC.Text:= UnitInfoList.Values['UIC'];

```

```

NotebookIni.Free;

```

```

UnitInfoList.Free;

```

```

{Fill Grid Labels}

```

```

Grid1.Cells[0,0]:= 'Officer Recall';

```

```

Grid1.Cells[0,1]:= 'CO/OIC';
Grid1.Cells[0,2]:= 'XO';
Grid1.Cells[0,3]:= 'OPS Officer';
Grid1.Cells[0,4]:= 'Maint, Officer';
Grid1.Cells[0,5]:= 'Flight Surgeon';
Grid1.Cells[0,6]:= 'ASO';
Grid1.Cells[0,7]:= 'Safety Officer';
Grid1.Cells[0,8]:= 'CACCO Officer';
Grid1.Cells[0,9]:= 'PAO Officer';

```

{Grid One Label fill}

```

Grid1.Cells[1,0]:= 'Rank&Name';
Grid1.Cells[2,0]:= 'Squadron';
Grid1.Cells[3,0]:= 'Home Phone';
Grid1.Cells[4,0]:= 'DNS Work Phone';

```

```

Grid2.Cells[0,0]:= 'Proposed Mishap Board';
Grid2.Cells[0,1]:= 'AMB Senior Member';
Grid2.Cells[0,2]:= 'Flight Surgeon';
Grid2.Cells[0,3]:= 'Maint. Officer';
Grid2.Cells[0,4]:= 'OPS Officer';
Grid2.Cells[0,5]:= 'Other';

```

```

Grid2.Cells[1,0]:= 'Rank&Name';
Grid2.Cells[2,0]:= 'Squadron';
Grid2.Cells[3,0]:= 'Home Phone';
Grid2.Cells[4,0]:= 'DNS Work Phone';

```

{fill Aircrew data label}

```

PersGrid.Cells[0,0]:= 'Position';
PersGrid.Cells[1,0]:= 'Rank';
if ServiceSel.ItemIndex = 1 then
  PersGrid.Cells[2,0]:= 'MOS' else
  PersGrid.Cells[2,0]:= 'Desg/NEC';
PersGrid.Cells[3,0]:= 'Service';
PersGrid.Cells[4,0]:= 'ParentUnit';
PersGrid.Cells[5,0]:= 'DutyStatus';
PersGrid.Cells[6,0]:= 'Sig. Injuries';
PersGrid.Cells[7,0]:= 'Hosp. Days';
PersGrid.Cells[8,0]:= 'LostWkDays';
PersGrid.Cells[9,0]:= 'NVGS USED';
PersGrid.Cells[10,0]:= 'TOTAL HOURS';
PersGrid.Cells[11,0]:= 'MODEL HOURS';
{Fill Aircraft Data Labels}

```

```

ACGrid.Cells[0,0]:='Model';
ACGrid.Cells[1,0]:='Bureau#';
ACGrid.Cells[2,0]:='Side#';
ACGrid.Cells[3,0]:='Unit';
ACGrid.Cells[4,0]:='EngineType/Model/Series';
ACGrid.Cells[5,0]:='EngineSerial#';
ACGrid.Cells[6,0]:='EquipmentModel';
ACGrid.Cells[7,0]:='EquipmentMake';
ACGrid.Cells[8,0]:='EquipmentPart#';
ACGrid.Cells[9,0]:='EquipmentCode#';
{FILL INJURED PASSENGER DATA FIELD}
PaxGrid.Cells[0,0]:='RANK';
if ServiceSel.ItemIndex = 1 then begin
  PaxGrid.Cells[1,0]:='MOS';
  PaxGrid.Cells[2,0]:='USMC';
end else begin
  PaxGrid.Cells[1,0]:='DESG.';
  PaxGrid.Cells[2,0]:='USN/DOD';
end;
PaxGrid.Cells[3,0]:='DOD/Non-DOD';
PaxGrid.Cells[4,0]:='Unit';
PaxGrid.Cells[5,0]:='Duty Status';
PaxGrid.Cells[6,0]:='InjuryType';
PaxGrid.Cells[7,0]:='InjuryDesc';
PaxGrid.Cells[8,0]:='HospitalDays';
PaxGrid.Cells[9,0]:='LostWorkDays';
{Create Intro Screen}
Intro:= TIntroForm.Create(self);
Intro.ShowModal;
if Intro.ModalResult = mrNo then Intro.Free
else begin
  Notebook.PageIndex:= 9;
  Intro.Free
end;

end; {End of MainForm Create}

{Officer recall update to the NoteBok.INI file
  Updates the officer recall list gridBox and the mishap
  board GridBox}
procedure TNoteBookForm.UpdateOfficerClick(Sender: TObject);
var
R, C: LongInt;

```



```

NoteBookIni: TIniFile;
begin
{Write Officer Recall to NoteBokIni}
if MessageDlg('Are you sure you want to change OFFICER RECALL
Information?',mtConfirmation,mbOKCancel,0) = mrOk
then begin
NoteBookIni:= TIniFile.Create('notebook.ini');
  for R:= 1 to Grid1.RowCount-1 do begin
    with NoteBookIni do begin
      WriteString('OfficerRecall','Name'+IntToStr(R), Grid1.Cells[1,R]);
      WriteString('OfficerRecall','UnitId' + IntToStr(R), Grid1.Cells[2,R]);
      WriteString('OfficerRecall','CPhone'+IntToStr(R), Grid1.Cells[3,R]);
      WriteString('OfficerRecall','APhone' +IntToStr(R), Grid1.Cells[4,R]);
    end;
  end;
  for R:= 1 to Grid2.RowCount-1 do begin
    with NoteBookIni do begin
      WriteString('MisBoard','Name'+IntToStr(R), Grid2.Cells[1,R]);
      WriteString('MisBoard','UnitId' + IntToStr(R), Grid2.Cells[2,R]);
      WriteString('MisBoard','CPhone'+IntToStr(R), Grid2.Cells[3,R]);
      WriteString('MisBoard','APhone' +IntToStr(R), Grid2.Cells[4,R]);
    end;
  end;
NoteBookIni.Free;
end;
end;

{Don't delete}
procedure TNoteBookForm.MishapCatDoneClick(Sender: TObject);
var
  MishapType, MishapInjury: TMishapClass;
begin
if (Damage.ItemIndex = 0) or (Damage.ItemIndex = 1)
then MishapType:= ClassA;
if (Damage.ItemIndex = 2) then MishapType:=ClassB;
if (Damage.ItemIndex = 3) then MishapType:=ClassC;

if (Injury.ItemIndex = 0) or (Injury.ItemIndex = 1)
then MishapInjury:= ClassA;
if (Injury.ItemIndex = 2) or (Injury.ItemIndex = 3)
then MishapInjury:=ClassB;
if (Injury.ItemIndex = 4) then MishapInjury:=ClassC;

```

```

if (MishapType = ClassA) or (MishapInjury = ClassA)
  then MishapSeverity.ItemIndex:= 0 else
if (MishapType = ClassB) or (MishapInjury = ClassB)
  then MishapSeverity.ItemIndex:= 1 else
if (MishapType = ClassC) or (MishapInjury = ClassC)
  then MishapSeverity.ItemIndex:= 2;

```

```

end;

```

```

{Determine mishap severity }
procedure TNoteBookForm.TabSet1Click(Sender: TObject);
begin
Notebook1.PageIndex:= TabSet1.TabIndex;
end;

```

```

procedure TNoteBookForm.Timer1Timer(Sender: TObject);
var
minutes:integer;
begin
Timer1.Tag:=Timer1.TAg+1;
Gauge1.Progress:= Timer1.Tag;
Label11.Caption:= IntToStr(Timer1.Tag) + ' Minutes of 60 into Mishap';
if (Timer1.Tag = 5) then
MessageDlg('OPREP-3 Voice Report Due. OPREP-3 Message due in 15
minutes',mtInformation, [mbOk], 0);
if (Timer1.Tag = 20)then
MessageDlg('OPREP-3 Message Due.',mtInformation, [mbOk], 0);
if (Timer1.Tag = 60)then
MessageDlg('Safety Center Voice Report Due.',mtInformation, [mbOk], 0);
if (Timer1.Tag = 30)then
MessageDlg('Mishap Report Due in 3.5 hours ',mtInformation, [mbOk], 0);
if (Timer1.Tag = 60)then
MessageDlg('Mishap Report Due in 3 hours ',mtInformation, [mbOk], 0);
if (Timer1.Tag = 120)then
MessageDlg('Mishap Report Due in 2 hours ',mtInformation, [mbOk], 0);
end;

```

```

procedure TNoteBookForm.StartTimerClick(Sender: TObject);
begin
Timer1.Enabled:=True;
end;

```

```

{Update Oprep Voice Report}
procedure TNoteBookForm.UpdateOprepClick(Sender: TObject);
var
Report: TStringList;
i: integer;
begin
  checkBox6.Checked:= True;
  Report:=TStringList.Create;
  OprepMemo.Lines.Clear;
  With Report do begin
  Add(DateAndTime.Text);
  Add(' ');
  Add(OprepPOC.Text +' THIS IS ' + SquadronName.Text + ' OPREP-3 ' +
  OprepType.Text + ' OVER. ');
  Add(' ');
  Add('Response:');
  Add(''+SquadronName.Text + ' THIS IS ' + OprepPOC.Text + ', SEND OPREP-3 ' +
  OprepType.Text + ' '+''');
  Add(' ');
  Add(OprepPOC.Text +' THIS IS ' + SquadronName.Text);
  Add(#13);
  If OprepType.ItemIndex = 0 then
    Add('FLASH') else Add('IMMEDIATE');
  Add('OPREP-3 ' + OprepType.Text);
  Add(' ');
  Add('UNCLASSIFIED');
  Add('LINE 1. N/A');
  Add('');
  Add('LINE 2: ' + MishapLocation.Text);
  Add('');
  Add('LINE 3: ' + MishapDescriptionMemo.Text);
  {filler so file will write to disk}
  for i:=1 to 30 do
    Add(' ');
  end;
  OprepMemo.Lines:= Report;
  Report.Free;

  end;
  {Synchronize POC PHone Number with phone list}
procedure TNoteBookForm.OprepPOCChange(Sender: TObject);
begin
POCPhone.ItemIndex:= OprepPOC.ItemIndex;

```

```

end;
{Update OPREP Message}
procedure TNoteBookForm.UpdateOPClick(Sender: TObject);
var
Report, TempList:TStringList;
i,j:integer;
begin
  Checkbox5.Checked:=True;
  Report:=TStringList.Create;
  MsgMemo.Lines.Clear;
  With Report do begin
    Add('PAAUZYUW JulianDate -UUUU--    ');
    Add('ZNR UUUUU');
    Add('P '+ DateAndTime.Text+ '  YZB');
    Add('');
    Add('FM: '+UnitPlad.Text);
    Add('TO: CNO WASHINGTON DC //JJJ//');
    Add('  '+ FleetCom.Text + '//JJJ//');
    Add('  '+ UniCom.text+ '//JJJ//');
    Add('  '+ TYCOM.Text+ '//JJJ//');
    Add('  '+ WingCom.Text + '//JJJ//');

    Add('INFO: ');
    Case Injury.ItemIndex of
      0: begin
        if (ServiceSel.ItemIndex = 1) then Add('  CMC WASHINGTON DC//JJJ//');
        Add('  CNO NOVEMBER ONE WASHINGTON DC //JJJ//');
        Add('  CHNAVPERS WASHINGTON DC //JJJ//');
        Add('  COMNISCOM WASHINGTON DC //22D//');
        end;
      1,2,3,4: begin
        if (ServiceSel.ItemIndex = 1) then Add('  CMC WASHINGTON DC//JJJ//');
        Add('  CNO NOVEMBER ONE WASHINGTON DC //JJJ//');
        Add('  CHNAVPERS WASHINGTON DC //JJJ//');
        end;
    end;
  end;
  {if DoDDDead.ItemIndex = 0 then begin}

  if CivDead.ItemIndex = 0 then Add('  NAVY JAG ALEXANDRIA VA');
  Add('  COMNAVAIRSYSCOM WASHINGTON DC//JJJ//');
  Add('  COMNAVSAFECEN NORFOLK VA //00/10/11/541//');
  Add('  NAVMARINTCEN WASHINGTON DC //JJJ//');

```

```

Add('BT');
Add('UNCLAS');
Add('MSGID/OPREP-3NB/'+SquadronName.Text+ '/00' +
IntToStr(MishapNumber.Value)+'/'
  + FormatDateTime('mmm',StrToDateTime(ZTime.Text)) + '//');
Add('REF/A/OPREP-3NB/'+SquadronName.Text + '/' + DateAndTime.Text + '//');
Add('FLAGWORD/'+ OPREPTYPE.Text + '/-//');
Add('TIMELOC/' + DateAndTime.Text+'/' + LatLong.Text + '/INIT//');
Add('GENTEXT/INCIDENT IDENTIFICATION AND DETAILS/');
Add(MishapLocation.Text);
Add(MishapDescriptionMemo.Text);
Add('MISHAP DATA. ');

Add('AIRCRAFT DATA. ');
TempList:=TStringList.Create;
TempList:=AddGridList(ACGrid);
AddStrings(TempList);
TempList.Free;

Add('CUSTODIAN LOCATION. '+ ShipPlad.text);
Add('MISSION. '+ MissionCode.Text);
Add('EVOLUTION. '+ AirEvol.Text);
Add('AIRCREW AND PASSENGERS. SOULS ONBOARD ' +
IntToStr(NumberCrew.Value+PaxNumber.Value));
Add('SAR STATUS. ');
Add('//');
Add("");
Add('RMKS/');
Add(OprepRemarks.Text+'//');
Add('BT');
Add(Grid1.Cells[1,1]+' , COMMANDING OFFICER, '+ SquadronName.Text);
end;
MsgMemo.Lines:= Report;
Report.Free;
end;

procedure TNoteBookForm.FormResize(Sender: TObject);
begin
Label11.Left:= Gauge1.Width div 2 - Label11.Width div 2;
end;
{CheckList Update }
procedure TNoteBookForm.UpdateUnitInfoClick(Sender: TObject);
var

```

```

NoteBookIni: TIniFile;
R,C:integer;
begin
if MessageDlg('Update Squadron Information?',mtConfirmation,mbOKCancel,0) = mrOk
then begin
NoteBookIni:= TIniFile.Create('notebook.ini');
  with NoteBookIni do begin
    WriteString('UnitInfo','UnitName',SquadronName.Text);
    WriteString('UnitInfo','UnitPlad',UnitPlad.Text);
    WriteString('UnitInfo','Address',SAddress.Text);
    WriteString('UnitInfo','State',SState.Text);
    WriteString('UnitInfo','Zip',SZip.Text);
    WriteString('UnitInfo','ShipPlad',ShipPlad.Text);
    WriteString('UnitInfo','AircraftType',AircraftTypeComboBox.Text);
    WriteString('UnitInfo','CADPlad',CADPlad.Text);
    WriteString('UnitInfo','AVPhone',AVPhone.Text);
    WriteString('UnitInfo','ComPhone',ComPhone.Text);
    WriteString('UnitInfo','Tycom',TYCOM.Text);
    WriteString('UnitInfo','WingCom',WingCom.Text);
    WriteInteger('UnitInfo','ZuluTimeSet',ZuluTimeSet.Value);
    WriteInteger('UnitInfo','MishapNumber',MishapNumber.Value);
    WriteInteger('UnitInfo','ServiceSel',ServiceSel.ItemIndex);
    WriteString('UnitInfo','UIC',UIC.Text);
    WriteString('UnitInfo','FleetCom',FleetCom.Text);
    WriteString('UnitInfo','UnifiedCom',UniCom.Text);
    for R := 1 to Grid1.RowCount - 1 do
      Grid1.Cells[2,R]:=SquadronName.Text;
    for R:= 1 to Grid2.RowCount -1 do
      Grid2.Cells[2,R]:=SquadronName.Text;

  end;
NotebookIni.Free;
Notebook.PageIndex :=1;
end;
end;
{Update Safety Center PHone Report}
procedure TNoteBookForm.UpdateSafetyCallClick(Sender: TObject);
var
Temp:string;
Report:TStringList;
i,j:integer;
begin
Report:=TStringList.Create;

```

```

With Report do begin
  Add('CALL: NAVAL SAFETY CENTER:');
  Add("");
  Add('  AUTOVON: 564-2929/3520');
  Add('  COMMERCIAL: (804)444-2929/3250 (CALL COLLECT)');
  Add("");
  Add('INCLUDE THE FOLLOWING INFORMATION:');
  Add(' ');
  Add('A: '+ SquadronName.Text);
  Add('B: '+ AircraftTypeComboBox.Text);
  Temp:="";
  for i:= ACGrid.RowCount downto 1 do
    Temp:= Temp+ ACGrid.Cells[1,i]+' ';
  Add('C: '+temp);
  Add('D: '+ MishapLocation.Text + ' The Lat/Long of the mishap is '+ LatLong.Text);
  Add('E: '+ MishapDescriptionMemo.Text);
  Add('F: '+ 'Dammage/Injury and Fatalities');
  Add('G: '+ Grid1.Cells[0,6]+ ': ' + Grid1.Cells[1,6]);
  Add(' COM: '+Grid1.Cells[3,6]+' AV: ' + Grid1.Cells[4,6] );
  Add("");
  end;
CheckBox8.Checked:=True;
SCVoice.Lines:= Report;
Report.Free;
end;
{Update MIshap Report}
procedure TNoteBookForm.UpdateMRClick(Sender: TObject);
var
  Temp:string;
  Report,TempList:TStringList;
  i,j,r,c:Integer;
begin
  CheckBox11.Checked:=True;
  Report:=TStringList.Create;
  MishapReport.Lines.Clear;
  With Report do begin
    Add('PAAUZYUW JulianDate -UUUU-- ');
    Add('ZNR UUUUU');
    Add('P '+ DateAndTime.Text+' YZB');
    Add("");
    Add('FM '+UnitPlad.Text);

    Add('TO CNO WASHINGTON DC//JJJ//');

```

```

Add('CMC WASHINGTON DC//A/SD//');
Add('COMNAVSAFECEN NORFOLK VA//00/10/11/FILE//');
Add(CadPlad.Text);

Add('INFO ');
Add('NAVAIRWARCENACDIV LAKEHURST NJ//JJJ//');
Add('COMNAVSEASYS COM WASHINGTON DC//JJJ//');
Add(ShipPlad.Text);
if DoDDead.ItemIndex = 0 then
  Add('ARMED FORCES INSTITUTE OF PATHOLOGY WASHINGTON
DC//CME-0//');
  if CivDead.ItemIndex = 0 then Add('NAVY JAG ALEXANDRIA VA //JJJ//');
  if CarLandBox.Checked then Add('LSO SCHOOL NAS OCEANA VA //JJJ//');
  if ALSSBox.Checked then begin
    Add('NAVAIRWARCENWPNDIV CHINA LAKE CA//JJJ//');
    Add('NAVAIRWARCENACDIV WARMINSTER PA//JJJ//');
    Add('ALL AEROMEDICAL ACTIVITIES//');
    end;

  if HeloLandBox.Checked then begin
    Add('HELSUPPRON EIGHT//');
    Add('HELSUPPRON THREE//');
    end;

  if SARCheckBox.Checked then Add('HELANTISUBRON ONE//60//');

Add('BT');
Add('UNCLAS FOUO //N03750//');
Add("");
Add('SUBJ/THIS IS AN INITIAL GENERAL USE NAVAL AIRCRAFT MISHAP
REPORT/');
Add(SquadronName.Text+' '+MisHapSeverity.Text+' '+MisHapCategory.Text+' '
+'0' + IntToStr(MishapNumber.Value)+' '
+ AnsiUpperCase(FormatDateTime('yy ',StrToDateTime(LTime.Text)))
+' '+ ACGrid.Cells[0,1] + ',/REPORT SYMBOL OPNAV 3750-20//');

Add('REF/A/DOC/OPNAVINST 3750.6Q/-//');
Add('REF/B/DOC/JAGINST 5800.7C/-//');
Add('RMKS/1. SUMMARY. '+ MishapDescriptionMemo.Text);
Add('2 DATA. ');
Add(' A. AIRCRAFT. ');
  for i:= 1 to ACGrid.RowCount -1 do begin
    Add(' ');
  
```



```

    for j:= 0 to ACGrid.ColCount - 1 do
        Add( ('+IntToStr(j+1)+' ) + ACGrid.Cells[j,i]);
    end;
Add("");
Add('B  EQUIPMENT. NA');
Add("");
Add('C. ENVIRONMENT. ');
Add('(1)+FormatDateTime('hhnn',StrToDateTime(LTime.Text)));
Add('(2)+FormatDateTime('ddmmyy',StrToDateTime(LTime.Text)));
Add('(3)+IntToStr(ZuluTimeSet.Value));
Temp:="";
if DayNiteSel.ItemIndex = 0 then Temp:= 'DAY' else Temp:='NIGHT';

Add('(4)+ Temp);
Add('(5)+LatLong.Text);
Add('(6)+Altitude.Text);
Add('(7)+Weather.Text);
Add("");
Add('3. CIRCUMSTANCES');
Add('(A)+Origin.Text);
Add('(B)+MissionCode.Text);
Add('(C)+FPC.Text);
Add('(D)+FlightPlan.Text);
Add('(E)+Destination.Text);
Add('(F)+AirEvol.Text);
Add("");
Add('4. MISHAP CATEGORY');
Add(MishapSeverity.Text+ ' '+MishapCategory.Text+ ' '
+ '0' + IntToStr(MishapNumber.Value)+ '- '
+ AnsiUpperCase(FormatDateTime('yy ',StrToDateTime(LTime.Text))));

Add("");
Add('5. DAMAGE AND COSTS');
Add("");
If Damage.ItemIndex= 0 then
    Add(' A. AIRCRAFT DESTROYED') else Add(' A. TBD');
Add(' B. TBD');
Add(' C. TBD');
Add("");
Add('6. PERSONNEL INFORMATION AND INJURIES');
Add(' A. SOULS ON BOARD: '+ IntToStr(NumberCrew.Value +
PaxNumber.Value));
Add(' B. CREW: '+IntToStr(NumberCrew.Value));

```

```

TempList:=TStringList.Create;
TempList:=AddGridList(PersGrid);
AddStrings(TempList);
TEmpList.Free;

Add(' ');
if PaxNumber.Value = 0 then begin
  Add('C. TOTAL NUMBER OF PASSENGERS: NA');
  Add('(1) INJURED PASSENGERS: NA');
  Add('(2) UNINJURED PASSENGERS: NA');
end else
  begin
    Add('C. TOTAL NUMBER OF PASSENGERS: '+
IntToStr(PaxNumber.Value));
    if InjuredPax.Value > 0 then
      begin
        Add('(1) INJURED PASSENGERS: '+IntToStr(InjuredPax.Value));
        TempList:=TStringList.Create;
        TempList:=AddGridList(PaxGrid);
        AddStrings(TempList);
        TempList.Free;
      end
    else Add(' (1) INJURED PASSENGERS: NA');
    Report.Add('(2) UNINJURED PASSENGERS: '+
IntToStr(PaxNumber.Value - InjuredPax.Value));
  end;

  Add('D. INJURED NON-OCCUPANTS: '+IntToStr(InjuredNonOccupants.Value));
  Add("");
  Add('E. AEROMEDICAL ANALYSIS WILL BE SENT');
  Add("");
  Add('7. MISHAP INVESTIGATION');
  Add("");
  Add('8. JAG MANUAL INVESTIGATION');
  Add("");
  Add(' THIS MISHAP (DOES/DOES NOT) MEET THE REQUIREMENTS IN REF
B FOR A JAG MANUAL INVESTIGATION');
  Add('9. POINTS OF CONTACT');
  Add("");
  TempList:=TStringList.Create;
  TempList:=AddGridList(Grid2);
  AddStrings(TempList);

```

```

TempList.Free;
Add('/');
Add('BT');
Add('#0001');
end;
MishapReport.Lines:= Report;
Report.Free;
end;

```

```

procedure TNoteBookForm.UpdateTimeClick(Sender: TObject);
var
LocalTimeStamp, ZuluTimeStamp: TDateTime;
begin
{Update Mishap Info Time Groups}
LocalTimeStamp:=Now;
ZuluTimeStamp:=(LocalTimeStamp - ZuluTimeSet.Value/24);
LTime.Text:=DateTimeToStr(LocalTimeStamp);
ZTime.Text:=DateTimeToStr(ZuluTimeStamp);
LocalTime.Text:=FormatDateTime('dd mmm yy hhnn',LocalTimeStamp);
ZuluTime.Text:=FormatDateTime('dd mmm yy hhnn',ZuluTimeStamp);
DateAndTime.Text:= FormatDateTime('ddhhnnZmmmyy',ZuluTimeStamp);
end;

```

```

procedure TNoteBookForm.UpdateMishapClick(Sender: TObject);
var
i,j,R: integer;
begin
{Fill Grids with test data}
for R:= 1 to PersGrid.RowCount-1 do begin
PersGrid.Cells[0,R]:='PAC';
if ServiceSel.ItemIndex = 1 then begin
PersGrid.Cells[1,R]:='Capt';
PersGrid.Cells[2,R]:='MOS';
PersGrid.Cells[3,R]:='USMC';
end else begin
PersGrid.Cells[1,R]:='LT';
PersGrid.Cells[2,R]:='13 10';
PersGrid.Cells[3,R]:='USN';
end;
PersGrid.Cells[4,R]:=SquadronName.Text;
PersGrid.Cells[5,R]:='ONDUTY';
PersGrid.Cells[6,R]:='NOINJURY';
PersGrid.Cells[7,R]:='#OFHOSP.';

```

```

    PersGrid.Cells[8,R]:='NOLWD';
    PersGrid.Cells[9,R]:='NVGS NOT USED';
    PersGrid.Cells[10,R]:='0HOURS';
    PersGrid.Cells[11,R]:='0HOURS';
    end;
end;

procedure TNoteBookForm.AircraftNumberChange(Sender: TObject);
begin
    ACGrid.RowCount:=AircraftNumber.Value + 1;
    ACGrid.ClientHeight:= Notebook1.Height - 50;
end;

procedure TNoteBookForm.NumberCrewChange(Sender: TObject);
begin
    if (NumberCrew.Value <=2) then begin
        PersGrid.Height:= 57;
        PersGrid.RowCount:=2;
    end;
    PersGrid.RowCount:= NumberCrew.Value + 1;
    PersGrid.ClientHeight:=Notebook1.Height - 75;
end;

procedure TNoteBookForm.About1Click(Sender: TObject);
var
    About: TAboutMishap;
begin
    About:=TAboutMishap.Create(NoteBookForm);
    About.ShowModal;
    About.Free;
end;

procedure TNoteBookForm.InjuredPaxChange(Sender: TObject);
begin
    if PaxNumber.Value < InjuredPax.Value then
        PaxNumber.Value:= PaxNumber.Value + 1;
    if (InjuredPax.Value <= 1) then begin
        PaxGrid.Height:= 57;
        PaxGrid.RowCount:=2;
    end else
        PaxGrid.RowCount:=InjuredPax.Value + 1;
    PaxGrid.ClientHeight:= Notebook1.Height - 75;
end;

```

```

procedure TNoteBookForm.UpdateMisAircraftClick(Sender: TObject);
var
i,j,k: integer;
begin
{Fill Aircraft Grid with default values}
for i := 1 to ACGrid.RowCount - 1 do
for j:=0 to ACGrid.ColCount -1 do begin
ACGrid.Cells[0,i]:= AircraftTypeComboBox.Text;
ACGrid.Cells[1,i]:='BUNO';
ACGrid.Cells[2,i]:='SIDE#';
ACGrid.Cells[3,i]:= SquadronName.Text;
ACGrid.Cells[4,i]:='TBD';
ACGrid.Cells[5,i]:='TBD';
end;
end;

procedure TNoteBookForm.Contents1Click(Sender: TObject);
begin
Application.HelpFile:=('notebook.hlp');
Application.HelpContext(1000);
end;

procedure TNoteBookForm.MishapReport1Click(Sender: TObject);
begin
SaveDialog.FileName:='misreprt.txt';
SaveDialog.Execute;
MishapReport.Lines.SaveToFile(SaveDialog.FileName);

end;

procedure TNoteBookForm.SafetyCenterVoiceReport1Click(Sender: TObject);
begin
SaveDialog.FileName:='centrvoc.txt';
SaveDialog.Execute;
SCVoice.Lines.SaveToFile(SaveDialog.FileName);

end;

procedure TNoteBookForm.SaveOprepVoiceReport1Click(Sender: TObject);
begin
SaveDialog.FileName:='Oprepvoc.txt';
SaveDialog.Execute;

```

```

MishapReport.Lines.SaveToFile(SaveDialog.FileName);
end;

procedure TNoteBookForm.OPREPMessage1Click(Sender: TObject);
begin
  SaveDialog.FileName:='OPREPmsg.txt';
  SaveDialog.Execute;
  MsgMemo.Lines.SaveToFile(SaveDialog.FileName);

end;

procedure TNoteBookForm.Button2Click(Sender: TObject);
var
  R:integer;
begin
  if PaxNumber.Value < InjuredPax.Value then
    PaxNumber.Value:= InjuredPax.Value;

  for R:= 1 to PAXGrid.RowCount-1 do begin
    if ServiceSel.ItemIndex = 0 then begin
      PaxGrid.Cells[0,R]:='RANK';
      PaxGrid.Cells[1,R]:='DESG';
      PaxGrid.Cells[2,R]:='USN';
    end else begin
      PaxGrid.Cells[0,R]:='RANK';
      PaxGrid.Cells[1,R]:='MOS';
      PaxGrid.Cells[2,R]:='USMC';
    end;
    PaxGrid.Cells[3,R]:='DOD';
    PaxGrid.Cells[4,R]:='TBD';
    PaxGrid.Cells[5,R]:='ONDUTY';
    PaxGrid.Cells[6,R]:='TBD';
    PaxGrid.Cells[7,R]:='TBD';
    PaxGrid.Cells[8,R]:='TBD';
    PaxGrid.Cells[9,R]:='TBD';
  end;
end;

procedure TNoteBookForm.AircraftTypeComboBoxChange(Sender: TObject);
begin
  CADPlad.ItemIndex:=AircraftTypeComboBox.ItemIndex;
end;

```

```

procedure TNoteBookForm.OprepVoiceReport1Click(Sender: TObject);
begin
    PrintReport(OprepMemo);
end;

procedure TNoteBookForm.OPREPMMessage2Click(Sender: TObject);
begin
    PrintReport(MsgMemo);
end;

procedure TNoteBookForm.SafetyCenterVoiceReport2Click(Sender: TObject);
begin
    PrintReport(SCVoice);
end;

procedure TNoteBookForm.MishapReport2Click(Sender: TObject);
begin
    PrintReport(MishapReport);
end;

procedure TNoteBookForm.TabContinueClick(Sender: TObject);
begin
    With Notebook do
        PageIndex:= PageIndex + 1;
end;

procedure TNoteBookForm.InfoCompClick(Sender: TObject);
begin
    Notebook1.PageIndex:= Notebook.PageIndex + 1;
    TabSet1.TabIndex:=TabSet1.TabIndex+1;
end;

procedure TNoteBookForm.MishapInfoCompleteClick(Sender: TObject);
begin
    if MessageDlg('MISHAP INFORMATION COMPLETE! The next step in the'+
        ' check list is to CREATE THE OPREP-3 MESSAGE.'+
        ' Do you want to continue with the Checklist?',
        mtConfirmation,mbOKCancel,0) = mrOk
    then begin
        CheckBox12.Checked:=True;
        Notebook.PageIndex:= 6;
    end;
end;

```

```

procedure TNoteBookForm.CompeteNextClick(Sender: TObject);
begin
if MessageDlg('OPREP-3 VOICE REPORT COMPLETE! The next step in the'+
' check list is FILL OUT THE REMAINING SUB TABS OF THE MISHAP INFO tab'+
'Do you want to continue with the Checklist?',
mtConfirmation,mbOKCancel,0) = mrOk
then begin
CheckBox7.Checked:= True;
NoteBook.PageIndex:=2;
NoteBook1.PageIndex:=1;
end;
end;

```

```

procedure TNoteBookForm.Button7Click(Sender: TObject);
begin
if MessageDlg('You have just completed the OPREP MESSAGE. The next step is '+
' is to create THE SAFETY CENTER VOICE REPORT.' +
' Do want to Continue?',
mtConfirmation,mbOKCancel,0) = mrOk
then begin
CheckBox6.Checked:= True;
TabContinue.Click;
end;
end;

```

```

procedure TNoteBookForm.Yes1Click(Sender: TObject);
begin
if (MisSevQ1.ItemIndex = 0) then MishapCategory.ItemIndex:= 4;

if (MisSevQ1.ItemIndex = 1) and (MisSevQ2.ItemIndex = 0) and
(MisSevQ3.ItemIndex = 0) and (MisSevQ4.ItemIndex = 0) and
(MisSevQ5.ItemIndex = 0) then MishapCategory.ItemIndex:= 3;

if (MisSevQ1.ItemIndex = 1) and (MisSevQ2.ItemIndex = 0) and
((MisSevQ3.ItemIndex = 1) or (MisSevQ4.ItemIndex = 1) or
(MisSevQ5.ItemIndex = 1)) then MishapCategory.ItemIndex:= 2;

if (MisSevQ1.ItemIndex = 1) and (MisSevQ2.ItemIndex = 1) and
(MisSevQ3.ItemIndex = 0) and ((MisSevQ4.ItemIndex = 1) or
(MisSevQ5.ItemIndex = 1)) then MishapCategory.ItemIndex:= 1;

if (MisSevQ1.ItemIndex = 1) and (MisSevQ2.ItemIndex = 1) and

```



```

(MisSevQ3.ItemIndex = 1) then MishapCategory.ItemIndex:= 0;

if (MisSevQ1.ItemIndex = 1) and (MisSevQ2.ItemIndex = 1) and
(MisSevQ3.ItemIndex = 0) and (MisSevQ4.ItemIndex = 0) and
(MisSevQ5.ItemIndex = 0) then MishapCategory.ItemIndex:= 3;
end;

procedure TNoteBookForm.Button6Click(Sender: TObject);
begin
if MessageDlg('OFFICER RECALL COMPLETE! The next step in the'+
' check list is to fill out PAGE ONE of the Mishap Info. Tab' +
' Do you want to continue with the Checklist?',
mtConfirmation,mbOKCancel,0) = mrOk
then begin
CheckBox5.Checked:=True;
TabContinue.Click;
end;
end;

procedure TNoteBookForm.GotoMishapCatClick(Sender: TObject);
begin
if MessageDlg('PAGE ONE of Mishap Info. COMPLETE! The next step in the'+
' check list is to determine Mishap Category.' +
' Do you want to continue with the Checklist?',
mtConfirmation,mbOKCancel,0) = mrOk
then begin
CheckBox9.Checked:= True;
TabContinue.Click;

end;
end;

procedure TNoteBookForm.CompleteMishapCatClick(Sender: TObject);
begin
if MessageDlg('MISHAP CATEGORY COMPLETE! The next step in the'+
' check list is to determine Mishap Severtiy.' +
' Do you want to continue with the Checklist?',
mtConfirmation,mbOKCancel,0) = mrOk
then begin
CheckBox3.Checked:= True;
TabContinue.Click;

end;
end;

```

end;

```
procedure TNoteBookForm.MishapSevDoneClick(Sender: TObject);
begin
if MessageDlg('MISHAP SEVERITY COMPLETE! The next step in the'+
' check list is to CREATE THE OPREP-3 VOICE REPORT'+
' Do you want to continue with the Checklist?',
mtConfirmation,mbOKCancel,0) = mrOk
then begin
CheckBox4.Checked:= True;
TabContinue.Click;
end;
end;
```

```
procedure TNoteBookForm.Button8Click(Sender: TObject);
begin
if MessageDlg('SAFETY CENTER VOICE REPORT COMPLETE! The next step in the'+
' checklist is to complete the MISHAP REPORT.'+
' Do want to Continue?',
mtConfirmation,mbOKCancel,0) = mrOk then begin
```

```
CheckBox8.Checked:= True;
TabContinue.Click;
end;
end;
```

```
procedure TNoteBookForm.MRCompleteClick(Sender: TObject);
begin
if MessageDlg('MISHAP REPORT COMLETE! You have finished all of the required'+
' steps in the checklist. Make sure you have saved and printed all' +
' applicable reports. Prssing OK will take you back to the CheckList Tab',
mtConfirmation,mbOKCancel,0) = mrOk then
begin
CheckBox11.Checked:=True;
NoteBook.PageIndex:=0;
end;
end;
```

```
procedure TNoteBookForm.OfficerRecall1Click(Sender: TObject);

begin
PrintGrid(Grid1,',');
```

```
PrintGrid(Grid2,');  
end;
```

```
end.
```

APPENDIX C

PROLOG CODE

A. PROLOG FACTS

```
/*File facts.pl
Created by Hemant Bhargava and Charles Emde on May 8, 1995.
Following our extensive discussions on how to formalize and
document
the mishap investigation process.
*/
/* This file contains the database of information inputted by the
user in the form of causes and supporting evidence. The evidence
is textual for clarity. The actual implementation should contain
database field pointers rather than the text descriptions here.*/
/* Description of Predicates Used:*/

/*
probable cause predicate (prob_cause(arg1,arg2)) is read as
"the probable cause of event(arg1) is event(arg2).

evidence predicate (evidence(prob_cause1,prob_cause2,supporting
evidence1))
is read as "supporting evidence1 implies that prob_cause2 is a
result of prob_cause1.
*/

/*Example: This mishap is fictional. It is the hypothetical
example taken from OPNAVINST 3750.6Q, Appendix M. The scenario
is described as a gear up landing. Please refer to the
instruction for details.
*/
/*root mishap*/

prob_cause(aircraft_mishap,gear_not_down).
evidence(aircraft_mishap,gear_not_down,wreckage_inspection).

/*one chain of events*/

prob_cause(gear_not_down,pilot_overlooked_indicators).
prob_cause(pilot_overlooked_indicators,fatigue).
prob_cause(fatigue,four_hours_sleep).
prob_cause(four_hours_sleep,overwork).
prob_cause(overwork,poor_supervision).
evidence(gear_not_down,pilot_overlooked_indicators,interview).
evidence(fatigue,four_hours_sleep,interview).
evidence(four_hours_sleep,overwork,interview).
/*there is no evidence to suggest poor supervision*/
```

```
/*a second chain of events based on the pilot overlooking the
indicators*/
```

```
prob_cause(pilot_overlooked_indicators,distracted).
prob_cause(pilot_distracted,father_died).
evidence(pilot_overlooked_indicators,distracted,interview).
```

```
/*There is no evidence that the distraction was caused by the
father's death.*/
```

```
/*a third chain of events*/
```

```
prob_cause(gear_not_down,handle_malfunctioned).
prob_cause(handle_malfunctioned,improper_maintenance).
prob_cause(improper_maintenance,omitted_step).
prob_cause(omitted_step,poorly_written_handbook).
prob_cause(poorly_written_handbook,missing_page).
evidence(gear_not_down,handle_malfunctioned,wreckage_inspection).
evidence(handle_malfunctioned,improper_maintenance,records_insp).
evidence(improper_maintenance,omitted_step,handbook).
evidence(omitted_step,poorly_written_handbook,expert_interview).
```

```
/*a fourth chain of events*/
```

```
prob_cause(gear_not_down,backup_not_used).
prob_cause(backup_not_used,poor_training).
prob_cause(poor_training,crew_uncoordinated).
prob_cause(crew_uncoordinated,no_coord_training).
prob_cause(no_coord_training,no_command_support).
evidence(gear_not_down,backup_not_used,wreckage_examination).
evidence(backup_not_used,poor_training,training_records).
evidence(poor_training,crew_uncoordinated,interview).
evidence(crew_uncoordinated,no_coord_training,training_records).
```

```
/*a fifth chain of events*/
```

```
prob_cause(gear_not_down,no_warning_horn).
prob_cause(no_warning_horn,malfunctioning_switch).
prob_cause(malfunctioning_switch,corrosion).
prob_cause(corrosion,improper_hanging).
prob_cause(improper_hanging,poor_supervision).
evidence(gear_not_down,no_warning_horn,wreckage_examination).
evidence(no_warning_horn,malfunctioning_switch,wreckage_examination).
evidence(malfunctioning_switch,corrosion,wreckage_examination).
evidence(corrosion,improper_hanging).
```

B. PROLOG RULES

```
/* File rules.pl
```

```
Created by Hemant Bhargava and Charles Emde on May 8, 1995.
Following our extensive discussions on how to formalize and document
the mishap investigation process.
```

```
*/
```

*/*rules: first rule states that a scenario consists of scenarios of at least two probable causes, and scenarios and events are transitive. /*

```
scenario(X, Y) :-  
    prob_cause(X,Y).
```

```
scenario(X, Y) :-  
    prob_cause(X,Z),  
    scenario(Z,Y).
```

/ A scenario chain is constructed by following ALL the events that are (transitively) probable causes for the event (X) in question. There may be multiple answers since an event may have more than one probable cause. There are two ways to do this: □ using recursion and using iteration (or accumulators).
/

/ The recursion solution shown below is elegant. However the problem is that it will create multiple answers including subsets of the correct chain.*

```
scenario_chain(Event,[Head|Rest]) :-  
    prob_cause(Event,Head),  
    scenario_chain(Head,Rest).  
scenario_chain(_,[]).
```

**/*

/ The iterative solution produces the same results, but does so in a constructive "forward chaining" approach, which consumes less memory. The second argument is the accumulator. At the end the accumulator has the answer: see the final clause. It is the final cause that prevents the multiple/subset answers from appearing as you ask for more solutions. */*

```
scenario_chain(Event, Chain) :-  
    scenario_chain_it(Event, [], Chain).
```

/ At the start you have accumulated nothing. */*

```
scenario_chain_it(Event, Temp, Chain) :-  
    prob_cause(Event, Cause), /* Now you add the first cause */  
    append(Temp, [Cause], Temp1), /* to your accumulator Temp1 */  
    scenario_chain_it(Cause, Temp1, Chain).
```

```
scenario_chain_it(Event, Chain, Answer) :-  
    (prob_cause(Event, _) , !, fail;  
    Answer = Chain).
```

/ When you reach an event which has no probable cause, you have accumulated in Chain the Answer for the original event. */*

/*The following allows the user view the results of his deliberation and returns the set of scenario explanations that are chained together and are supported by evidence*/

```
explain_scenario(X,Y,Explanation) :-  
    prob_cause(X,Y),  
    evidence(X,Y,EV),  
    Explanation = [X,'was caused by',Y,'and is supported by',EV,'...'].
```

```
explain_scenario(X,Y,Explanation) :- not(prob_cause(X,Y)),  
    prob_cause(X,Z),  
    explain_scenario(X,Z,ExpXZ),  
    scenario(Z,Y),  
    explain_scenario(Z,Y,ExpZY),  
    append(ExpXZ,ExpZY,Explanation).
```

C. EXAMPLE SCRIPTS

1. prob_cause(X,Y).

```
prob_cause(X,Y).
```

```
X = aircraft_mishap
```

```
Y = gear_not_down ;
```

```
X = gear_not_down
```

```
Y = pilot_overlooked_indicators ;
```

```
X = pilot_overlooked_indicators
```

```
Y = fatigue ;
```

```
X = fatigue
```

```
Y = four_hours_sleep ;
```

```
X = four_hours_sleep
```

```
Y = overwork ;
```

```
X = overwork
```

```
Y = poor_supervision ;
```

```
X = pilot_overlooked_indicators
```

Y = distracted ;

X = pilot_distracted

Y = father_died ;

X = gear_not_down

Y = handle_malfunctioned ;

X = handle_malfunctioned

Y = improper_maintenance ;

X = improper_maintenance

Y = omitted_step ;

X = omitted_step

Y = poorly_written_handbook ;

X = poorly_written_handbook

Y = missing_page ;

X = gear_not_down

Y = backup_not_used ;

X = backup_not_used

Y = poor_training ;

X = poor_training

Y = crew_uncoordinated ;

X = crew_uncoordinated

Y = no_coord_training ;

X = no_coord_training

Y = no_command_support ;

X = gear_not_down
Y = no_warning_horn ;

X = no_warning_horn
Y = malfunctioning_switch ;

X = malfunctioning_switch
Y = corrosion ;

X = corrosion
Y = improper_hanging ;

X = improper_hanging
Y = poor_supervision ;

No

2. scenario_chain(X,Y).

scenario_chain(X,Y).

X = aircraft_mishap
Y = [gear_not_down,pilot_overlooked_indicators,fatigue,
four_hours_sleep,overwork,poor_supervision] ;

X = aircraft_mishap
Y = [gear_not_down,pilot_overlooked_indicators,distracted] ;

X = aircraft_mishap
Y = [gear_not_down,handle_malfunctioned,improper_maintenance,
omitted_step,poorly_written_handbook,missing_page] ;

X = aircraft_mishap
Y = [gear_not_down,backup_not_used,poor_training,crew_uncoordinated,
no_coord_training,no_command_support] ;

X = aircraft_mishap

Y = [gear_not_down,no_warning_horn,malfunctioning_switch,corrosion,
improper_hanging,poor_supervision] ;

X = gear_not_down

Y = [pilot_overlooked_indicators,fatigue,four_hours_sleep,
overwork,poor_supervision] ;

X = gear_not_down

Y = [pilot_overlooked_indicators,distracted] ;

X = pilot_overlooked_indicators

Y = [fatigue,four_hours_sleep,overwork,poor_supervision] ;

X = fatigue

Y = [four_hours_sleep,overwork,poor_supervision] ;

X = four_hours_sleep

Y = [overwork,poor_supervision] ;

X = overwork

Y = [poor_supervision] ;

X = pilot_overlooked_indicators

Y = [distracted] ;

X = pilot_distracted

Y = [father_died] ;

X = gear_not_down

Y = [handle_malfunctioned,improper_maintenance,omitted_step,
poorly_written_handbook,missing_page] ;

X = handle_malfunctioned

Y = [improper_maintenance,omitted_step,poorly_written_handbook,
missing_page] ;

X = improper_maintenance

Y = [omitted_step,poorly_written_handbook,missing_page] ;

X = omitted_step

Y = [poorly_written_handbook,missing_page] ;

X = poorly_written_handbook

Y = [missing_page] ;

X = gear_not_down

Y = [backup_not_used,poor_training,crew_uncoordinated,no_coord_training,
no_command_support] ;

X = backup_not_used

Y = [poor_training,crew_uncoordinated,no_coord_training,
no_command_support] ;

X = poor_training

Y = [crew_uncoordinated,no_coord_training,no_command_support] ;

X = crew_uncoordinated

Y = [no_coord_training,no_command_support] ;

X = no_coord_training

Y = [no_command_support] ;

X = gear_not_down

Y = [no_warning_horn,malfunctioning_switch,corrosion,improper_hanging,
poor_supervision] ;

X = no_warning_horn

Y = [malfunctioning_switch,corrosion,improper_hanging,poor_supervision] ;

X = malfunctioning_switch

Y = [corrosion,improper_hanging,poor_supervision] ;

X = corrosion

Y = [improper_hanging,poor_supervision] ;

X = improper_hanging

Y = [poor_supervision] ;

No

D. INTERFACE TOOLS

1. Query Generation

queryexe :-

```
/* scenario_chain_exe, */
```

```
scenario_exe.
```

scenario_exe :-

```
/* repeat, */
```

```
show_scenario(X),
```

```
scenario(X,Y),
```

```
write(X), write(' caused by '), write(Y).
```

scenario_exe. /* when there's no more to go. */

scenario_chain_exe(Event) :-

```
scenario_chain(Event, Answer),
```

```
write(Answer).
```

```
/*
```

scenario_chain_exe :-

```
repeat,
```

```
show_scenario(Event), comes from query file
```

```
scenario_chain(Event, Answer),
```

write(Answer), modify this to write answer nicely fail.* /

APPENDIX D

MISHAP EXPERT VERSION 1.01B OBJECT PASCAL SOURCE CODE

A. MAIN FORM

```
unit Main;
interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Menus, TabNotBk, Strngfun, Allsc,
  Explain, Misexp, Unsupp;

type
  TMainForm = class(TForm)
    MainMenu: TMainMenu;
    FileNewItem: TMenuItem;
    FileOpenItem: TMenuItem;
    FileSaveItem: TMenuItem;
    FileSaveAsItem: TMenuItem;
    FilePrintItem: TMenuItem;
    FilePrintSetupItem: TMenuItem;
    FileExitItem: TMenuItem;
    EditUndoItem: TMenuItem;
    EditCutItem: TMenuItem;
    EditCopyItem: TMenuItem;
    EditPasteItem: TMenuItem;
    WindowTileItem: TMenuItem;
    WindowCascadeItem: TMenuItem;
    WindowArrangeItem: TMenuItem;
    HelpContentsItem: TMenuItem;
    HelpSearchItem: TMenuItem;
    HelpHowToUseItem: TMenuItem;
    HelpAboutItem: TMenuItem;
    OpenFileDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    PrintDialog: TPrintDialog;
    PrintSetupDialog: TPrinterSetupDialog;
    SpeedBar: TPanel;
    SpeedButton1: TSpeedButton; { &New }
    SpeedButton2: TSpeedButton; { &Open... }
    SpeedButton3: TSpeedButton; { &Save }
    SpeedButton4: TSpeedButton; { Save &As... }
    SpeedButton5: TSpeedButton; { &Print... }
    SpeedButton6: TSpeedButton; { P&rint Setup... }
    SpeedButton7: TSpeedButton; { &Undo }
    SpeedButton8: TSpeedButton; { Cu&t }
    SpeedButton9: TSpeedButton; { &Copy }
```

```

SpeedButton10: TSpeedButton; { &Paste }
SpeedButton11: TSpeedButton;
TabbedNotebook1: TTabbedNotebook;
Panel1: TPanel;
Label1: TLabel;
Label2: TLabel;
Panel2: TPanel;
Label3: TLabel;
Button1: TButton;
Label4: TLabel;
Memo1: TMemo;
Panel3: TPanel;
Edit3: TEdit;
Memo2: TMemo;
Memo3: TMemo;
ListBox2: TListBox;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Button2: TButton;
Panel4: TPanel;
Label8: TLabel;
ListBox3: TListBox;
ListBox4: TListBox;
Label9: TLabel;
Button3: TButton;
Label10: TLabel;
Label12: TLabel;
Button4: TButton;
Button5: TButton;
Edit1: TEdit;
Edit2: TEdit;
ListBox1: TListBox;
Label11: TLabel;
Panel5: TPanel;
Label13: TLabel;
ListBox5: TListBox;
Panel6: TPanel;
Label14: TLabel;
ListBox6: TListBox; { &Contents }
procedure FileNew(Sender: TObject);
procedure FileOpen(Sender: TObject);
procedure FileSave(Sender: TObject);
procedure FileSaveAs(Sender: TObject);
procedure FilePrint(Sender: TObject);
procedure FilePrintSetup(Sender: TObject);
procedure FileExit(Sender: TObject);
procedure EditUndo(Sender: TObject);
procedure EditCut(Sender: TObject);
procedure EditCopy(Sender: TObject);
procedure EditPaste(Sender: TObject);
procedure WindowTile(Sender: TObject);

```

```

procedure WindowCascade(Sender: TObject);
procedure WindowArrange(Sender: TObject);
procedure HelpContents(Sender: TObject);
procedure HelpSearch(Sender: TObject);
procedure HelpHowToUse(Sender: TObject);
procedure HelpAbout(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Edit1Enter(Sender: TObject);
procedure Edit1Exit(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
procedure Edit2KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
procedure Edit2Exit(Sender: TObject);
procedure Edit3KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
procedure Edit3Exit(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
end;

```

var

```

MainForm: TMainForm;
Scenario: TStringList;
EvList: TStringList;
TempList:TStringList;
ChainList:TStringList;
ChainIndex:Integer;
{TempCauseList: TStringList;}

```

implementation

{SR *.DFM}

```

procedure TMainForm.FileNew(Sender: TObject);
begin
  { Add code to create a new file }
end;

```

```

procedure TMainForm.FileOpen(Sender: TObject);
begin
  if OpenFileDialog.Execute then
  begin
    { Add code to open OpenFileDialog.FileName }
  end;
end;

```

```

procedure TMainForm.FileSave(Sender: TObject);
begin

```



```

    { Add code to save current file under current name }
end;

procedure TMainForm.FileSaveAs(Sender: TObject);
begin
    if SaveDialog.Execute then
        begin
            { Add code to save current file under SaveDialog.FileName }
            end;
        end;

procedure TMainForm.FilePrint(Sender: TObject);
begin
    if PrintDialog.Execute then
        begin
            { Add code to print current file }
            end;
        end;

procedure TMainForm.FilePrintSetup(Sender: TObject);
begin
    PrintSetupDialog.Execute;
end;

procedure TMainForm.FileExit(Sender: TObject);
begin
    Close;
end;

procedure TMainForm.EditUndo(Sender: TObject);
begin
    { Add code to perform Edit Undo }
end;

procedure TMainForm.EditCut(Sender: TObject);
begin
    { Add code to perform Edit Cut }
end;

procedure TMainForm.EditCopy(Sender: TObject);
begin
    { Add code to perform Edit Copy }
end;

procedure TMainForm.EditPaste(Sender: TObject);
begin
    { Add code to perform Edit Paste }
end;

procedure TMainForm.WindowTile(Sender: TObject);
begin
    Tile;
end;

```

```

end;

procedure TMainForm.WindowCascade(Sender: TObject);
begin
  Cascade;
end;

procedure TMainForm.WindowArrange(Sender: TObject);
begin
  Arrangelcons;
end;

procedure TMainForm.HelpContents(Sender: TObject);
begin
  Application.HelpCommand(HELP_CONTENTS, 0);
end;

procedure TMainForm.HelpSearch(Sender: TObject);
const
  EmptyString: PChar = "";
begin
  Application.HelpCommand(HELP_PARTIALKEY, Longint(EmptyString));
end;

procedure TMainForm.HelpHowToUse(Sender: TObject);
begin
  Application.HelpCommand(HELP_HELPONHELP, 0);
end;

procedure TMainForm.HelpAbout(Sender: TObject);
begin
  { Add code to show program's About Box }
end;

procedure TMainForm.Button1Click(Sender: TObject);
var
  i, j : integer;
begin
  for i := 0 to ListBox1.Items.Count - 2 do
    begin
      Scenario.Add(ProbCause(ListBox1.Items[i], ListBox1.Items[i+1]));
      ListBox3.Items := Scenario;
      ListBox6.Items := Scenario;
    end;
  end;

procedure TMainForm.Edit1Enter(Sender: TObject);
begin
  Scenario:=TStringList.Create;
  EvList:=TStringList.Create;
  {TempCauseList.Create;}
end;

```

```

procedure TMainForm.Edit1Exit(Sender: TObject);
begin
  Button1.Enabled := True;
  ListBox1.Items.Add(Edit1.Text);
  edit1.enabled := false;
  label1.visible := false;
end;

procedure TMainForm.Button2Click(Sender: TObject);
begin
  ListBox2.Items.Add(Edit3.Text);
  ListBox4.Items.Add(Edit3.Text);
  Edit3.Clear;
  {EvList.Add(Edit3.Text);
  ListBox2.Items:=EvList;
  ListBox4.Items:=EvList;
  Edit3.Clear;
  Edit3.SetFocus;}
end;

procedure TMainForm.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if (key = VK_RETURN) then
    Edit2.SetFocus;
end;

procedure TMainForm.Edit2KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if (key = VK_RETURN) then
    panel1.setfocus;
    edit2.setfocus;
end;

procedure TMainForm.Edit2Exit(Sender: TObject);
begin
  ListBox1.Items.Add(Edit2.Text);
  Edit1.Text:=Edit2.Text;
  Edit1.Enabled:=False;
  Edit2.Clear;

  label1.visible := false;
  label11.visible := true;
end;

procedure TMainForm.Edit3KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if (key = VK_RETURN) then

```

```

Memo2.setFocus;
end;

procedure TMainForm.Edit3Exit(Sender: TObject);
begin

    ListBox2.Items.Add(Edit3.Text);
    ListBox4.Items.Add(Edit3.Text);
    Edit3.Clear;
end;

procedure TMainForm.Button3Click(Sender: TObject);

var k,l:integer;
    first, second:string;
begin
    try
        first:=ListBox3.Items[ListBox3.ItemIndex];
        Delete(first,1,11);
        Delete(first,Length(first)-1,2);
        ListBox5.Items.Add('evidence('+first+', '+ListBox4.Items[ListBox4.ItemIndex]+
        ')');
    except
        on E: EStringListError do
            MessageDlg('Select a prob_cause and evidence!', mtInformation,
            [mbOK], 0);
    end;

end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
    ChainList:=TstringList.Create;
    TempList:=TStringList.Create;
    ChainIndex := 0;

end;

procedure TMainForm.Button5Click(Sender: TObject);
begin
    TempList.AddStrings(ListBox5.Items);
    TempList.Addstrings(ListBox6.Items);
    ChainList.AddObject(IntToStr(ChainIndex), TempList);
    ChainIndex:=ChainIndex + 1;
    TempList:=TStringList(ChainList.Objects[0]);
    TempList.SaveToFile('c:\pl\facts.pl');

    BtnRightDlg.Show;

end;

end.

```

B. UNSUPPORTED SCENARIO SCREEN

```
unit Unsup;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls, Buttons,
    StdCtrls, ExtCtrls;

type
  TBtnRightDlg2 = class(TForm)
    OKBtn: TBitBtn;
    CancelBtn: TBitBtn;
    HelpBtn: TBitBtn;
    Bevel1: TBevel;
    Panel4: TPanel;
    Label8: TLabel;
    ListBox3: TListBox;
    Label1: TLabel;
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    ListBox1: TListBox;
    procedure Button1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  BtnRightDlg2: TBtnRightDlg2;

implementation
  uses Main;
  {$R *.DFM}
  procedure TBtnRightDlg2.Button1Click(Sender: TObject);

var
  Prolog:word;
  FileName: string;
begin
  Prolog:=WinExec('c:/pl/pl -f c:/pl/start.pl',1);
  ListBox1.Items.LoadFromFile('c:\pl\Answer.pl');
end;

procedure TBtnRightDlg2.FormActivate(Sender: TObject);
begin
  listBox3.Items := TempList;
end;

end.
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145 2
2. Library, Code 013
Naval Postgraduate School
Monterey, CA 93943-5101 2
3. Professor Bhargava , Code SM/BH
Naval Postgraduate School
Monterey, CA 93943-5101 1
4. Professor Segumpta, Code SM/KS
Naval Postgraduate School
Monterey, CA 93943-5101 1
5. Aviation Safety Programs, Code 034 Attn.: MAJor Tom Hazzard, USMC
Naval Postgraduate School
1588 Cunningham Rd., RM 301
Monterey, CA 93943-5202 1
6. LT Hugh Brien
8661 Point of Woods Dr
Manassas, VA 22110 1
7. LT Charles Emde
165B Ocean View Blvd.
Pacific Grove, CA 93950 2