NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEVELOPMENT AND ANALYSIS OF AN EXPERT SYSTEM AND INTELLIGENT SOFTWARE AGENT FOR AVIATION SAFETY ASSESSMENT

by

Thomas R. Flowers
David M. Dowler

March 2000

Thesis Advisor: C. Thomas Wu
Thesis Co-Advisor: Anthony Ciavarelli

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2000	Master's Thesis

4. TITLE AND SUBTITLE: Development of An Expert System and Software Agent for Aviation Safety Assessment	5. FUNDING NUMBERS
6. AUTHOR(S) Flowers, Thomas R.; Dowler, David M.	
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 authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited.	

13. ABSTRACT *(maximum 200 words)*

The primary goal of this thesis is to design, develop and test an internet based prototype model for using expert system and software agent technologies to automate some of the analytical tasks in conducting aviation safety assessments using the data collected by the automated Aviation Command Safety Assessment (ACSA) system.

The Aviation Command Safety Assessment is a questionnaire survey methodology developed to evaluate a Naval Aviation Command's safety climate, culture, and safety program effectiveness. The survey was a manual process first administered in the fall of 1996. The survey was then automated in 1999 and is administered over the World Wide Web.

The results of this thesis are a prototype model and a software agent application that evaluates data contained in the ACSA database for organizational safety assessment and for database integrity. All source code is provided and discussed in detail.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Database, JDBC, Java, Expert Systems, Software Agents, Aviation Safety	158
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

Approved for public release; distribution is unlimited

DEVELOPMENT OF AN EXPERT SYSTEM AND SOFTWARE AGENT FOR AVIATION SAFETY ASSESSMENT

Thomas R. Flowers
Captain, United States Army
B.A., University of Texas, 1990

David M. Dowler
Lieutenant, United States Navy
B.S., University of Washington, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

ABSTRACT

The primary goal of this thesis is to design, develop and test an internet based prototype model for using expert system and software agent technologies to automate some of the analytical tasks in conducting aviation safety assessments using the data collected by the automated Aviation Command Safety Assessment (ACSA) system.

The Aviation Command Safety Assessment is a questionnaire survey methodology developed to evaluate a Naval Aviation Command's safety climate, culture, and safety program effectiveness. The survey was a manual process first administered in the fall of 1996. The survey was then automated in 1999 and is administered over the World Wide Web.

The results of this thesis are a prototype model and a software agent application that evaluates data contained in the ACSA database for organizational safety assessment and for database integrity. All source code is provided and discussed in detail.

TABLE OF CONTENTS

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to develop a prototype software system that will assist in the analysis and maintenance of the Aviation Command Safety Assessment database, which is maintained at the Naval Postgraduate School in Monterey, California. The database contains answers and respondent information from Aviation Command Safety Assessment (ACSA) surveys, which are administered over the World Wide Web. The goal of this thesis is to design, implement, test, and analyze a simple model that uses automated reasoning and software agent technologies to assist aviation safety researchers in recognizing patterns and identifying anomalies in the database. The secondary goal of the thesis is to develop the system using internet technologies, such as, Java, JDBC, Hypertext Transfer Protocol (HTTP), JavaScript, Active Server Pages (ASP), Java Servlets, Transmission Control Protocol (TCP), Internet Protocol (IP), and Hypertext Markup Language (HTML).

B. BACKGROUND

The United States Navy aircraft loss rate has steadily declined over the past twenty years from ten aircraft losses per 100,000 flight hours to a current level of approximately two aircraft losses for every 100,000 flight hours. Despite this decline, the number of aircraft losses due to human error has remained relatively constant.

About 60 percent of naval aircraft mishaps that cause death, permanent disability, or over one million dollars in losses are caused by aircrew errors. Aircrew errors, however, are often not isolated to just the crew involved. Mishaps usually involve a chain of events, many of which, begin at the organizational level. In 1996, the Commander Naval Air Pacific chartered a Human Factors Quality Management Board (QMB), which recognized that a squadron's safety climate was a critical link in the mishap chain of events. The board appointed Dr. Anthony Ciavarelli of the Naval

Postgraduate Safety School as the principal investigator for measuring command safety climate throughout naval aviation.

Dr. Ciavarelli and his research team (LTCOL Robert Figlock, USMC, Dr. Karlene Roberts and her associates from UC Berkeley and UCLA) developed the Aviation Command Safety Assessment (ACSA) survey to evaluate the effectiveness of a command's safety climate, culture, and safety program. The ACSA has been automated and is administered as a web application over the public Internet. The automated ACSA has enabled the QMB to quickly and efficiently survey and simultaneously evaluate communities throughout naval aviation.

The automated ACSA Survey System is accessible on the public internet as shown in Figure 1 and Figure 2. The website enables survey respondents to answer survey questions from anywhere in the world. Respondents answer survey questions with responses that range from from strongly disagree to strongly agree.

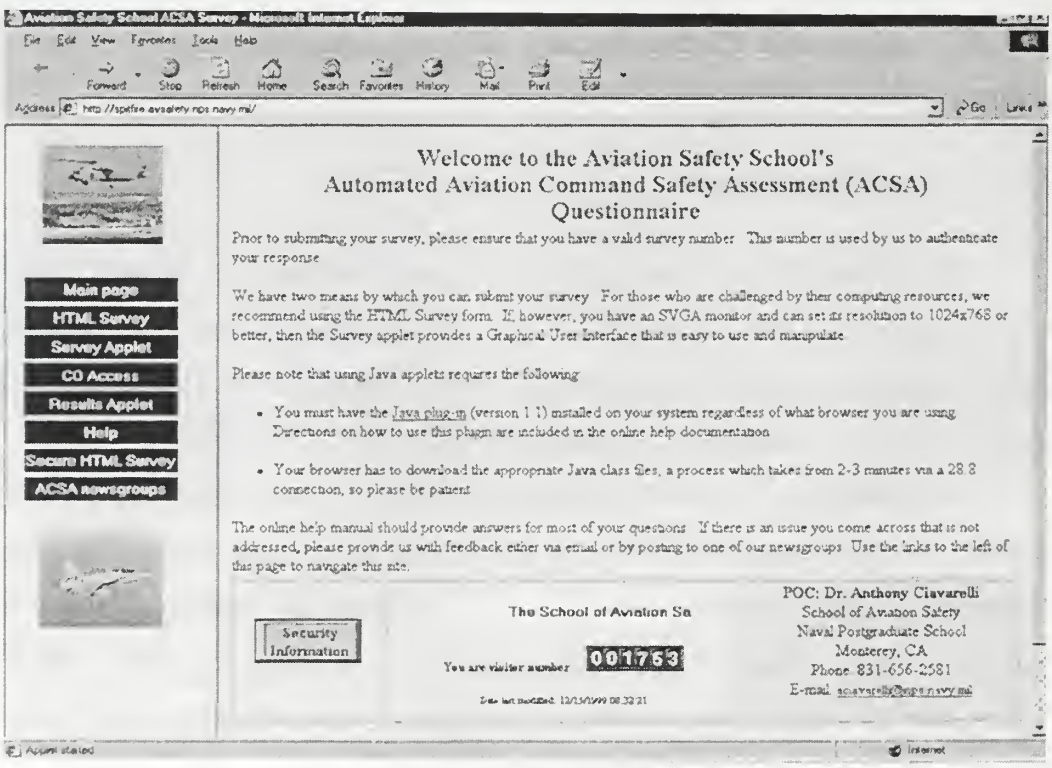

Figure 1: ACSA Home Page

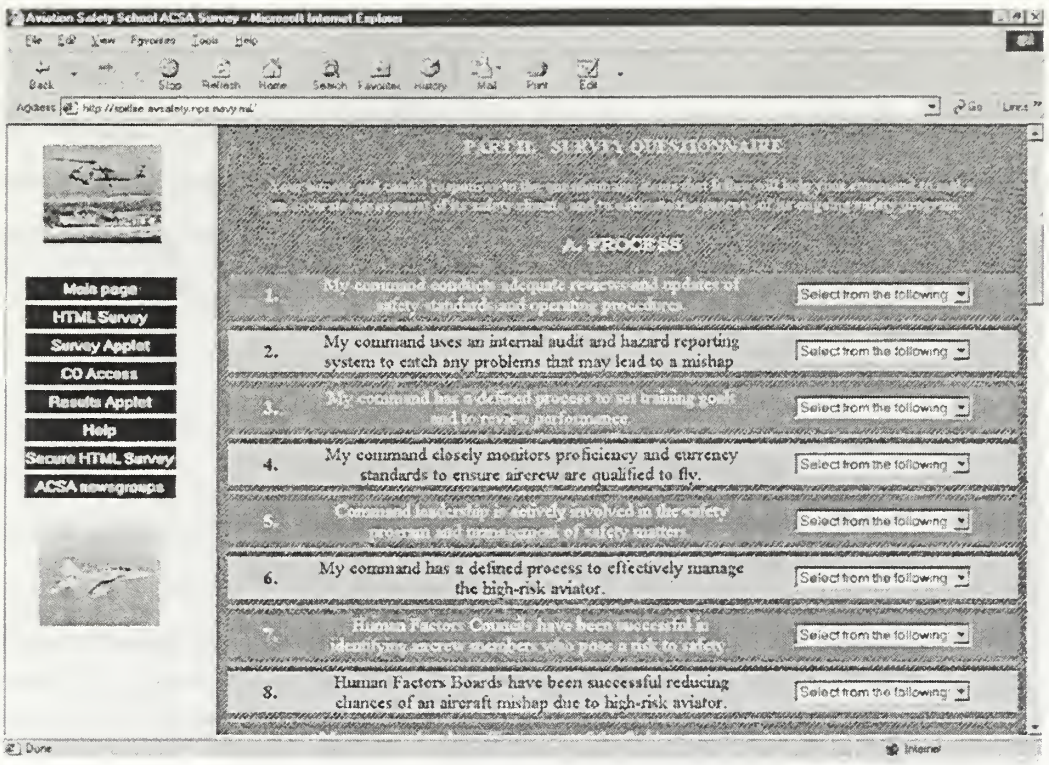

Figure 2: ACSA Survey Form

Respondent answers and some demographic data concerning the respondent are then stored in a relational database management system (RDBMS), Microsoft SQL Server 7.0, located at the Naval Postgraduate School.

This thesis takes the automated ACSA system one step further by developing an automated tool for data retrieval, analysis, and action. Artificial intelligence, expert system, database, and internet technologies are available for developing software agents that can automatically query data from the database, find patterns, match the patterns to rules generated by the human subject matter expert, and then execute an action as specified by the user. Actions could include printing a report, sending an email notification, or displaying an alert message on the screen.

Additional development of the ACSA system includes improving the web application that administers the survey. The original system uses a database design that is inherently inflexible and susceptible to corruption. This thesis designs, tests, and

implements a new database design that corrects many of the database problems of the original design. The new design also dramatically extends the capabilities of the system for administering the survey and for conducting research. A commercial internet applications developer, Universal Internet, Inc., has begun upgrading the automated ACSA survey system for the Naval Postgraduate School. The new database design sits as the core information system of the new ACSA web site. Chapter III of this thesis discusses the new database design.

C. THESIS ORGANIZATION

This thesis is organized into the following chapters:

- Chapter I: Introduction. This chapter provides an introduction for the thesis and the reason for research in the specific topic area.

- Chapter II: ACSA. This chapter describes the ACSA survey in more detail, and it explains the rationale and the model, upon which, the survey was constructed. The chapter also covers the technical aspects of the automated ACSA Survey System, as it was originally developed.

- Chapter III: Database Design. Core to any information system is its database, and this chapter explains our redesign of the ACSA database and its ramifications.

- Chapter IV: Expert Systems. This chapter provides a broad discussion of the current issues and technology concerning expert systems.

- Chapter V: Software Agents This chapter discusses the current technologies and applications of software agents.

- Chapter VI: The Design and Operation of the ACSA Agent. This chapter provides a detailed explanation of the development, design, analysis, and implementation of the software system.

D. PRODUCTS

The final products of this thesis include a model for developing automated software assistants and an application that demonstrates how the model may be used. The application, which is called the ACSA Safety Agent, runs as a Java application and

can connect to the ACSA database over a network. The application creates a software agent that periodically analyzes the database and produces reports on its findings. The agent produces five organizational safety assessments and three database integrity checks. The assessments and integrity analysis are summarized into two different reports, which may be automatically emailed to the user, saved to disk, displayed on the screen, or sent to a printer for printing. Chapter VI discusses the model and the application in further detail.

THIS PAGE INTENTIONALLY LEFT BLANK

II. THE AVIATION COMMAND SAFETY ASSESSMENT SURVEY

A. BACKGROUND

1. Organizational Safety Effectiveness Model

Accidents are not always simply the result of errors made by a pilot, a co-pilot, a navigator, or an aircraft mechanic. The so-called "human error" accident is often the tragic conclusion of a list of antecedents that are not necessarily in the hands of the system operator or maintainer. Accident causation often includes the linkage between both active failures (those that are caused by the operator or maintainer) and latent failures (those that are caused by organizational strategies). One approach to preventing active failures is to develop a system that identifies latent failures, which can ultimately lead to an active failure. The Organizational Safety Effectiveness Model identifies the characteristics and functions of an organization that are related to an organization's safety effectiveness. The model and the Aviation Command Safety Assessment (ACSA) Survey were developed to provide diagnostic data about an organization's risk for accidents.

Dr. Ciavarelli and Robert Figlock, both from the Naval Postgraduate School, developed the following conceptual framework, which was adapted from a model developed by Dr. Carolyn Libuser from UCLA and by Dr. Karlene Roberts from the Haas Business School, UC Berkeley. The model includes five categories related to organizational safety effectiveness:

- **Process Auditing** - A system of ongoing checks to identify hazards and to correct safety problems.

- **Reward System** - The expected social rewards and disciplinary actions used to reinforce safe behavior and to correct unsafe behavior.

- **Quality Control** - The policies and the procedures for promoting high quality of work performance.

- **Risk Management** - A systematic process used to identify hazards and to control operational risks. Includes accurate risk assessments.

- **Command and Control** - Reflects the organization's overall safety climate, leadership effectiveness, and the policies and procedures used in the management of flight operations and safety.

2. The ACSA Survey Questionnaire

Questions were developed to elicit measurable data concerning the five safety effectiveness model areas. Statistical tests were conducted to obtain measurement reliability of the survey questionnaire and to determine the factorial composition of the survey questionnaire. The analyses indicated an average reliability value of 0.80 across the five safety effectiveness model areas. The survey questions are answered according to a Likert scale: strongly disagree, moderately disagree, somewhat disagree, neutral, somewhat agree, moderately agree, strongly agree, and not applicable. The questions are described in Table 1.

An earlier version of the survey was administered in 1996 by mail to 69 randomly selected navy and marine squadrons. Since that time, however, an updated version of the survey has been automated and is accessible via the World Wide Web.

Table 1: ACSA Survey Questions

Process Auditing	
No.	**Question Text**
1	My command conducts adequate reviews and updates of safety standards and operating procedures.
2	My command uses an internal audit and hazard reporting system to catch any problems that may lead to a mishap.
3	My command has a defined process to set training goals and to review performance.
4	My command closely monitors proficiency and currency standards to ensure aircrew are qualified to fly.
5	Command leadership is actively involved in the safety program and the management of safety matters.
6	My command has a defined process to effectively manage the high-risk aviator.
7	Human Factors Councils have been successful in identifying aircrew members who pose a risk to safety.

8	Human Factors Boards have been successful in reducing chances of an aircraft mishap due to the high-risk aviator.
9	My command makes effective use of the flight surgeon to help identify and manage high-risk personnel.
Reward System and Safety Culture	
10	Command leadership encourages reporting safety discrepancies without the fear of negative repercussions.
11	Individuals in my command are willing to report safety violations, unsafe behaviors, or hazardous conditions.
12	In my command, peer influence is effective at discouraging violations of standard operating procedures, or safety rules.
13	In my command, we believe safety is an integral part of flight operations.
14	In my command, anyone who intentionally violates standard procedures, or safety rules, is swiftly corrected.
15	In my command, violations of operating procedures, flying regulations, or general flight discipline are rare.
16	Leaders in my command encourage everyone to be safety conscious and to follow the rules.
17	In my command, an aviator who persistently violates flight standards and rules will seriously jeopardize his/her career.
18	I am not comfortable reporting a safety violation because people in my command would react negatively toward me.
Quality	
19	My command has a reputation for high quality performance.
20	My command sets high quality standards and strives to maintain quality control.
21	My command closely monitors quality and corrects any deviations from established quality standards.
22	Quality standards in my command are clearly stated in formal publications and procedural guides.
Risk Management	
23	Command leaders permit cutting corners to get a job done.
24	Lack of experienced personnel has adversely affected my command's ability to operate safely.
25	Safety decisions are made at the proper levels, by the most qualified people in my command.
26	Command leaders consider safety issues during the formation and execution of operational and training plans.
27	Command leadership has a clear picture of the risks associated with its flight operations.

28	My command takes the time to identify and assess risks associated with its flight operations.
29	My command does a good job managing risks associated with its flight operations.
30	My command has increased the chances of a mishap due to inadequate or incorrect risk assessment.
31	I am provided adequate resources (time, staffing, budget, and equipment) to accomplish my job.
32	My command provides the right number of flight hours per month for me to fly safely.
33	I have adequate time to prepare for my brief and flight.
34	Based upon my command's personnel and other assets, the command is over-committed.
35	My command has incorporated Operational Risk Management processes in decision making at all levels.
Command and Control	
36	My supervisor can be relied on to keep his/her word.
37	Our command leaders and supervisors can be trusted.
38	My command's Safety Officer is highly regarded.
39	Our Safety Officer is influential in promoting safety.
40	My command is genuinely concerned about safety.
41	Command leadership is successful in communicating its safety goals to unit personnel.
42	My command provides a positive command climate that promotes safe flight operations.
43	Command leadership is actively involved in the safety program and management of safety matters.
44	Command leadership sets the example for compliance with flight standards.
45	My command ensures that all unit members are responsible and accountable for safe flight operations.
46	Command leadership willingly assists in providing advice concerning safety matters.
47	Command leadership reacts well to unexpected changes to its plans.
48	My command does not hesitate to temporarily restrict from flying individuals who are under high personal stress.
49	I am adequately trained to safely conduct all of my flights.
50	Morale and motivation in my command are high.

51	My command ensures the uniform enforcement of all operating standards among unit members.
52	Crew rest and standards are enforced at my command.
53	In my command, NATOPS test and check rides are conducted as intended, to candidly assess aircrew qualifications.
54	My command provides adequate safety backups to catch possible human errors during high-risk missions.
55	Within my command, good communications flow exists up and down the chain of command.
56	My command has good two-way communication with external commands.
57	Safety education and training are adequate in my command.
58	The Safety Department is a well-respected element of my command.
59	The Aviation Safety Officer position is a sought-after billet in my command.
60	My command's Safety Department keeps me well informed regarding important safety information.
61	My command's Aircrew Coordination Training Program is helping to improve mission performance and safety.

B. THE ACSA SURVEY SYSTEM ON THE WORLD WIDE WEB

In 1999, Fred Mingo and John Held, who were graduate students at the Naval Postgraduate School, developed and implemented the automated ACSA Survey System. The system consists of a password-protected website, a backend database, and a survey number assignment and validation application. The website, as shown in Figure 1, is located at http://spitfire.avsafety.nps.navy.mil. Users, who have a valid user ID and password, can access three basic applications from the ACSA website:

- **Survey Number and Squadron ID Administration** - The system administrator uses this application to assign squadron identification numbers and to assign valid survey numbers to squadrons. The data is entered into a database, which is checked later for valid survey numbers and squadron IDs whenever a respondent attempts to submit a completed survey via the automated ACSA Survey System. Access to this application requires an additional password for system administrators only.

- **The Automated ACSA Survey System** - After a squadron's aviation safety officer (ASO) has contacted the Aviation Safety School at the Naval Postgraduate School, and has requested that the survey be administered to his or her squadron, the system administrator gives the ASO a unique squadron ID and a block of valid survey numbers. The ASO then administers the survey to his or her squadron by distributing the survey numbers to the members of the squadron. Squadron members then may go to the ACSA website and complete the survey. The system will only accept a completed survey if the respondent enters a valid squadron ID coupled with a valid survey number. The data is then stored in a relational database system, Microsoft SQL Server 7.0, at the Naval Postgraduate School. Survey responses are kept completely anonymously and cannot be associated to a respondent.

- **The Commander's Module** - Only squadron commanders can access this application. The Commander's Module provides the commander with comparative charts, which graphically display average results by question by community, service, and location.

C. TECHNICAL DISCUSSION

1. Database Design and Implementation

Developers designed the ACSA database to model the original pen and ink survey form, which resulted in a relational database schema that is only in first normal form (1NF). The entity relation diagram is displayed in Figure 3, and the schema is explained in Table 2 and Table 3. The inherent weaknesses of the database design will be discussed in the next chapter, Chapter III: Database Design and Implementation. This section will present the details of the design and implementation only.

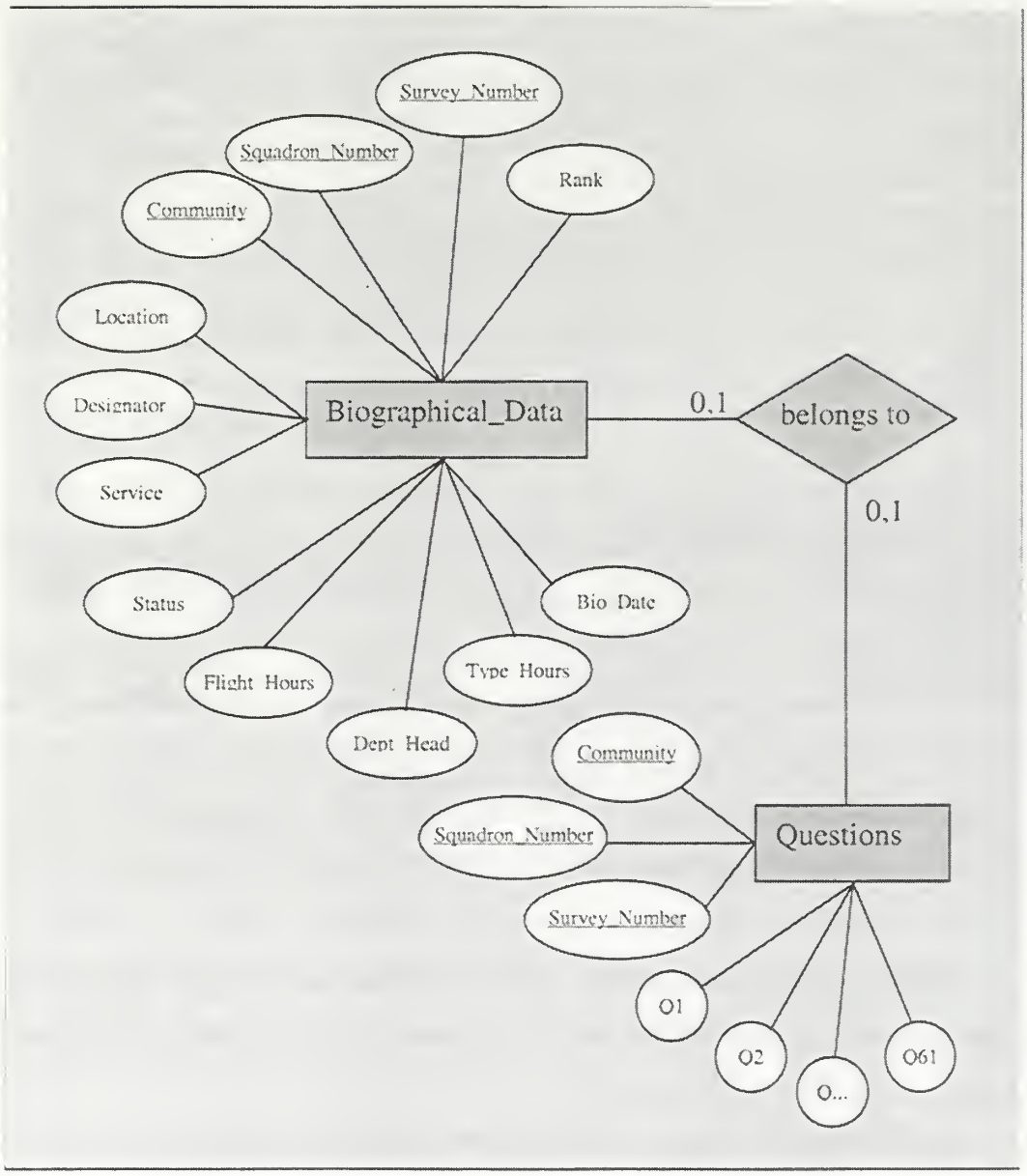

Figure 3: ER Diagram for Original Database

Table 2: Biographical_Data Table Schema

Field Name	Type	Table Constraints
Community	VARCHAR(50)	Primary Key
Squadron_Number	INTEGER	Primary Key
Survey_Number	INTEGER	Primary Key
Rank	INTEGER	
Designator	VARCHAR(50)	
Flight_Hours	INTEGER	
Type_Hours	INTEGER	
Dept_Head	VARCHAR(50)	
Status	VARCHAR(50)	
Service	VARCHAR(50)	
Location	VARCHAR(50)	
Bio_Date	VARCHAR(50)	

Table 3: Questions Table Schema

Field Name	Type	Table Constraints
Community	VARCHAR(50)	Primary Key
Squadron_Number	INTEGER	Primary Key
Survey_Number	INTEGER	Primary Key
Q1	INTEGER	
Q2	INTEGER	
Q3	INTEGER	
Q4...Q60	INTEGER	
Q61	INTEGER	

Developers implemented the database in Microsoft Access and then used an Access add-in to automatically migrate the database to SQL Server 6.5. The DBMS has since been upgraded to SQL Server 7.0.

An additional database was created for tracking valid survey numbers. The Num_List_DB database contains one table called Num_List_DB, and its schema is shown in Table 4. The system administrator uses Num_List_DB for assigning a block of numbers to a Squadron_ID. The Squadron_ID is a concatenation of values from the domain for Community and the domain for Squadron_Number. For example, a Unit_ID

HMM902 is eventually entered by a respondent into the Community field (HMM) and the Squadron_Number field (902) of the ACSA Survey form. The system then inserts the two fields into the ACSA database tables when the respondent completes the survey. When a user submits a survey, the system checks to ensure that the respondent's Survey_Number is within the block of integers designated by the Start_Number and End_Number fields in the Num_List_DB table.

Table 4: The Num_List_DB Schema

Field Name	Type	Table Constraints
Unit_ID	VARCHAR(50)	
Start_Number	INTEGER	
End_Number	INTEGER	

2. Website Implementation

As explained earlier in Chapter II, Section B, the website offers three applications: the ACSA Survey, the Administrator Tool, and the Commander's Module. The website, the databases, and the applications are hosted on a 200MHz Pentium PC running NT Server and Microsoft Internet Information Server (IIS). The applications use a mix of programming languages and web application methodologies.

The ACSA Survey runs both as a Java applet and as an HTML form-based application. The applet is rarely used, so we will not discuss its implementation. It uses Java 2, which requires an initial download of the Java 1.2 Plugin. Instead, users prefer to use the HTML form-based application. The survey application uses HTML and Javasript for data entry and validation on the front-end. The survey submits its data to a Java servlet, which is running on the same server as the web server. The servlet connects to the database and enters the data into the ACSA database via Java JDBC.

The Administrator Tool runs as an Active Server Page (ASP) application, which offers a webpage front-end form for entering and querying data in the Num_List_DB database. The Commander's Module is an applet that connects to the database via Java JDBC.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DATABASE DESIGN

A. INTRODUCTION

System developers used the pen and ink survey form as a model for developing the original ACSA database. While effective in providing a persistent storage medium for form data, the database design methodology that the developers followed did not produce a database that adheres to relational database theory. Therefore, the relational database management system (RDBMS) cannot reliably apply relational algebra operations and relational calculus to the database. The database also does not prevent insertion and deletion anomalies, since the database is not normalized. Effectively, the original database is unreliable and inflexible. Such weaknesses would need to be eliminated in order to successfully develop an intelligent software agent for database analysis. This chapter will explain how the database was re-designed to support current and future applications. First, this chapter briefly introduces the concept of enforcing business rules in database design. Second, this chapter discusses the weaknesses and problems of the current database. Finally, this chapter discusses the database design process that was followed in order to produce a new database that eliminates or mitigates the problems of the current database.

B. INTEGRITY CONSTRAINTS IN DATA MODELING

A database stores information about some part of the real world, which is called the miniworld or the universe of discourse. Rules, which are called business rules or integrity constraints, govern the universe of discourse. Business rules include domain constraints, key constraints, relationship constraints, and general semantic integrity constraints. When designing a database, the developer must identify the business rules that the database must enforce. The developer should specify as many of these rules as possible to the RDBMS and make the RDBMS responsible for enforcing them. [Ref. 1]

C. ACSA DATABASE WEAKNESSES AND PROBLEMS

1. Corrupted Data

The RDBMS enforces virtually none of the business rules governing the ACSA database. For example, the Automated ACSA Survey System depends upon at least three separate databases, between which, the RDBMS enforces no relational constraints. Survey answers and respondent data are maintained in one database, squadron numbers and survey IDs are maintained in another database, and CO passwords are maintained in a third database. The RDBMS enforces no relationships between these databases; although, relationships obviously do exist between several of the entities. For example, a survey ID has a relationship to a respondent, because respondents are given access to the survey site via their assigned survey ID number. The database, however, neither expresses nor enforces the relationship. The relationship is enforced by the application software, instead. In fact, the system relies almost solely upon the application software that runs the Automated ACSA Survey System to enforce all business rules, including, domain constraints.

Many functional dependencies are also enforced at the application level instead of at the database management level; while, in other cases, functional dependencies are totally ignored. For example, the respondent's survey ID functionally determines his or her squadron number and community. This functional dependency is expressed as ***survey ID → squadron number, community***. The RDBMS does not enforce either of these functional dependencies. Instead, the application enforces ***survey ID → squadron number***, while ***survey ID → community*** is not enforced at all.

The placement of virtually all business rules at the application level has resulted in a database that can inadvertently be corrupted if the database is accessed from an access point other than the application that enforces the rules. For example, a user may access the database from the database's own user interface or from another application that doesn't enforce the rules. The RDBMS provides little protection in preserving the meaning of the data under the original design. Therefore, an intelligent software agent

cannot rely upon the information contained within the original ACSA database. In fact, the database exhibits signs of corruption, which are discussed further below.

2. Inflexibility

Another major problem of the current ACSA database concerns inflexibility. The current design does not allow users to add, remove, or modify survey questions without risking loss of information. Such changes also require significant changes to the current Automated ACSA Survey System application software. This is caused by the fact that the database does not contain any information about a question as an entity in its own right. The database only contains information about the answers.

3. Ambiguous Data

The intelligent software agent must transform individual data cells, aggregate data, and data patterns into meaningful information, so that the agent can make conclusions based upon a set of rules and explicit and inferred facts contained in the database. The current design hinders the safety agent in two ways. First, null values are allowed in all fields, other than the primary key fields. From a database design and administration point of view, allowing null fields is often a pragmatic design decision. The information may not be known when a record is entered into the database, or the data is not critical to operations but is kept for secondary reasons. Null values, however, should be avoided when possible, because null values can lead to problems with understanding the meaning of the attributes and when specifying join operations at the logical level [Ref. 1]. In the case of the ACSA survey, no pragmatic reason exists for allowing null values in any field of the database. Therefore, nulls should not be allowed whatsoever. Missing information will degrade the overall effectiveness of the intelligent agent and result in inferior conclusions, unless the agent can ascertain why the information is missing. A null value can mean either that the field does not apply to this record, or that the value for the field is unknown, or that the value is known but is absent. Which of the three meanings is correct is anybody's guess.

Another problem concerning ambiguous data has been caused by the survey itself and has no relation to the database design. It is mentioned here briefly since it is related

to ambiguous data. One of the possible answers to any survey question is "NA." Respondents have been using "NA" to mean "I do not know", "I don't understand the question", "This question is not applicable to me", or "This question is not applicable to my unit." The survey designers have taken steps recently to refine the answer scale on the survey to be more specific.

4. Examples of Database Anomalies

The following database anomalies were found after thoroughly reviewing the original database:

- Six records in the Biographical_Data table do not contain values for "Designator" and "Dept_Head". These fields should not be empty, since they both apply to all respondents and since the application enforces rules that are supposed to prevent blank respondent fields from being entered into the database.

- Two of the eleven records for Squadron HMM924 indicate "Other" as their location. The other nine records indicate "West Coast." This is a result of the functional dependency ***survey ID*** $\rightarrow$ ***community*** not being enforced by the RDBMS.

- There are four records in the Biographical_Data table, for which, no corresponding records in the Questions table exist. The database either contains records for people who never took the survey or contains records for people whose survey answers have been deleted. No conclusion can be drawn from this information. Furthermore, the application software supposedly would not allow this difference between table records to occur. Also, since the database was modeled after the survey form, it appears that something is amiss if the Biographical_Data and Questions tables contain unequal numbers of records.

D. REQUIREMENTS FOR A NEW DATABASE

With the inherent weaknesses of the original database exposed, it became necessary in the development of the intelligent agent to re-design the database and populate the new database with valid information from the original database. The design process followed a complete database design process, which included requirements collection and analysis, conceptual database design, logical database design, and physical database design. Furthermore, the design was presented to commercial contractors, who

are currently developing a production-level Automated ACSA Survey System for the Naval Postgraduate School, for their review and design consideration. The first step in the design process was to make a list of requirements for the new database design. Below is the list of all the requirements that were to be addressed during the development of the new database.

1. Problem Statement

Requirements analysis began with the development a problem statement that describes the system's data requirements plain text English:

> The Naval Postgraduate School administers the Aviation Safety Command Assessment Survey via the World Wide Web. Respondent biographical data and survey answers are stored electronically in a relational database after the respondent has completed all fields of the survey, has selected the submit button, and has entered a valid survey identification number. The database must support the following use case and data requirements:
>
> Squadrons are assigned a block of survey IDs. Survey IDs are then distributed to squadron members who then complete the ACSA Survey on-line using their Squadron ID and Survey ID as validation. Squadrons may or may not use all of their assigned survey IDs. A respondent may only fill out the survey once using a Squadron ID and Survey ID pair. After the respondent completes the survey, the Survey ID for that squadron is no longer valid. ACSA administrators can add or remove questions at any time. If a question is deleted, then all of the answers for that question are deleted. If a question ID is changed, then all the answers to that question will reflect the new question ID. ACSA Administrators will be able to maintain multiple versions of the questionnaire in the database, and they will be able to track the answers to a question by questionnaire version. If a squadron is deleted from the database, then all Survey IDs, respondent information, and answers related to that squadron are deleted from the database. Squadrons must have a community and a location. Respondents must submit data for all biographical fields. The date (month, day, year) that the respondent completed the survey will be recorded. The database may only contain respondent information and survey answers that belong to a valid Survey ID. ACSA administrators must be able to view the database longitudinally by the respondent's data, by squadrons, by community, by location, by date, by question, and by questionnaire version.

2. Confidentiality and Anonymity Constraints

One of the goals for the ACSA survey system is to ensure that respondents and squadrons do not feel that the system is being used for evaluation purposes, but rather, it is strictly being used as a tool for identifying possible safety problems that could lead to aviation mishaps. As a result, the database design is constrained by the following requirements:

- Respondents are guaranteed anonymity.
- Squadrons are guaranteed confidentiality.

3. Respondent Information

In order for the agent to draw conclusions based on different geographical, as well as professional demographics, the database will collect and store the following data concerning respondents:

Table 5: Respondent Data Requirements

Data Name	Description of the Data
Rank	Possible values are: "E1-E5", "E6-E9","O1-O3","O4-O6"
Designator	Possible values are: "NFO", "Pilot", "Aircrew"
Flight Hours	A number x, where x is the total number of flight hours for the respondent and $0 \leq x \leq 25{,}000$
Type Hours	A number y, where y is the number of flight hours the respondent has for the type of aircraft he or she currently flies and $0 \leq y \leq x$
Squadron	The squadron, to which, the respondent is assigned
Status	The respondent's active duty / reserve status. Possible values are: "Regular", "Active Reserve", "Drilling Reserve"
Service	The respondent's service component. Possible values are: "Navy", "USMC", "Other"
Department Head?	Is the respondent a department head? Yes or no.
Date	When did the respondent complete the survey?
Aircraft Type	What type of aircraft does the aviator fly?

4. Survey Questions and Answers

In order to create the ability to change, add, and delete questions in a survey, it is necessary for the database to contain both the questions and the answers. This also provides for the ability of the software application to create objects dynamically, and allows for changes to the questions database without degrading database information integrity. The database must store the following information about survey answers:

Table 6: Question And Answer Data Requirements

Data Name	Description of the Data
Question Number	A number x, where $1 \leq x \leq$ the number of survey questions
Question Text	The question
Answer	A number y, where y is an answer given by a respondent for question number x and $-1 \leq y \leq 6$
Respondent	Who does this answer belong to?
Safety Area	What organizational safety model area does this question belong to? Possible values are: "Process Auditing", "Reward System", "Quality", "Risk Management", "Organization and Control"
Positive?	Is the question positively worded, which means that to agree to the statement is a positive answer.

5. Squadrons

The database must store the following information about squadrons:

Table 7: Squadron Data Requirements

Data Name	Description of the Data
Squadron Identification	The identification of the squadron
Community	The community, to which, the squadron belongs. Possible values are: "HC", "HCS", "HM", "HMH", "HMT","HS", "HSC", "HSL", "VAQ", "VAW", "VF", "VFA", "VMAQ", "VMFA", "VP", "VQ", "VR", "VRC", "VS", "VT", "VX", "Other"

Location	The location of the squadron. Possible values are: "East Coast", "West Coast", and "Other"

6. Questionnaires

Users must be able to change the survey without crippling software applications that depend upon the database. Applications should be able to dynamically adjust to survey modifications. Therefore, the database should contain information that describes the survey. In particular, the database must store the following information about surveys:

Table 8: Questionnaire Data Requirements

Data Name	Description of the Data
Questionnaire Version	The identification of the questionnaire
Update Date	When was the questionnaire version last updated
Questions	What questions belong to this questionnaire and what order are they in the questionnaire?

7. Flexibility

Aviation Safety School personnel can change survey questions, question numbers, and the order and grouping of questions. Such changes will not require fields to be added or removed to or from the database, and such changes will not require the database to be re-designed. This will also allow the database to archive old questions from older versions of questionnaires and their answers. Archiving the old questions and answers will provide good tools for future improvements to the intelligent agent, as well as ensure that database integrity remains intact despite new modifications.

The database will support software development, so that that any software that depends upon the database for information will not need modification if questions are added, removed, or modified.

E. CONCEPTUAL DESIGN

The first step in the conceptual design phase is to develop an entity-relation (ER) diagram that depicts the entity types, the attributes, and the relationships between the entity types. The ER diagram below represents the initial conceptual design that was developed during conceptual design phase of the database design process. Compare this ER diagram to the ER diagram in Chapter 2 that reflects the conceptual design of the original ACSA database.

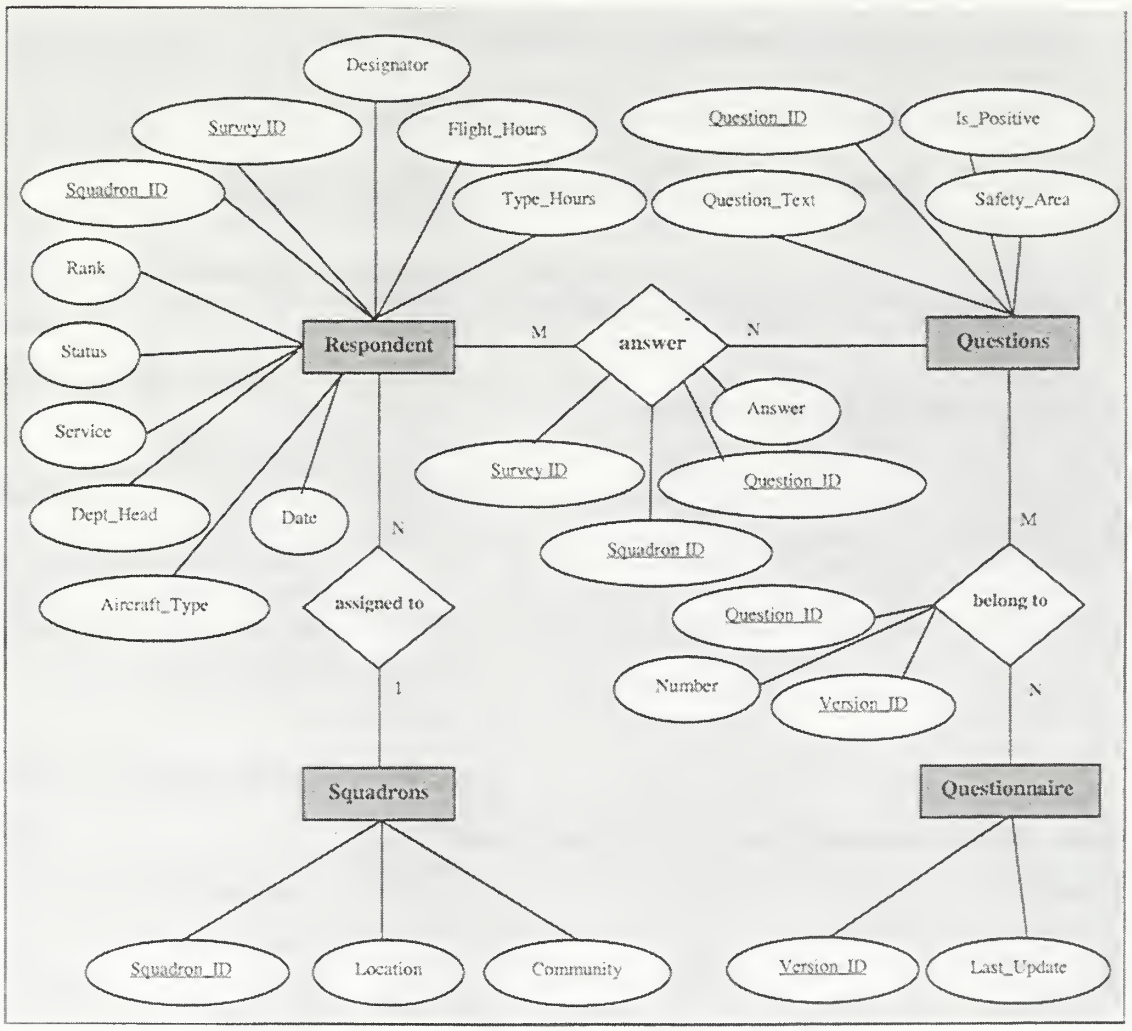

Figure 4 : Redesigned ER Diagram

F. LOGICAL DESIGN AND TABLE SCHEMATA

The following table schemata and tables describe a logical design that maps the ER diagram to a relational design that is in Boyce-Codd normal form (BCNF):

- **Squadrons** (Squadron_ID, Community, Location)
- **Squadrons_Survey_IDs** (Survey_ID, Squadron_ID)
- **Respondents** (Survey_ID, Squadron_ID, Date, Rank, Designator, Flight_Hours, Type_Hours, Dept_Head, Status, Service, Aircraft_Type)
- **Answers** (Survey_ID, Squadron_ID, Question_ID, Answer)
- **Questions** (Question_ID, Question_Text, Safety_Area, Is_Positive)
- **Questions_Questionnaires** (Questionnaire_Version, Question_ID, Question_Number)
- **Questionnaires** (Questionnaire_Version, Last_Update)

Table 9 : Squadrons Table Schema

Field	Type	Constraints
Squadron_ID	INT	Primary Key
Community	VARCHAR(25)	Not null, cascade deletes and updates
Location	VARCHAR(25)	Not null, cascade deletes and updates

Table 10: Squadrons_Survey_IDs Table Schema

Field	Type	Constraints
Survey_ID	INT	Primary Key
Squadron_ID	INT	Primary Key, references Squadrons

Table 11: Questionnaires Table Schema

Field	Type	Constraints
Questionnaire_Version	INT	Primary Key
Last_Update	DATE	Not null

Table 12: Questions Table Schema

Field	Type	Constraints
Question_ID	INT	Primary Key
Question_Text	VARCHAR(255)	Not null
Is_Positive	BIT(1)	Default B '1', not null
Safety_Area	VARCHAR(50)	Not null

Table 13: Questions_Questionnaires Table Schema

Field	Type	Constraints
Questionnaire_Version	INT	Primary Key, references Questionnaires, cascade deletes and updates
Question_ID	INT	Primary Key, references Questions, cascade deletes and updates
Question_Number	INT	Not null

Table 14: Respondents Table Schema

Field	Type	Constraints
Survey_ID	INT	Primary Key; references Squadrons_Survey_IDs, cascade deletes and updates
Squadron_ID	INT	Primary Key; references Squadrons_Survey_IDs, cascade deletes and updates
Date	DATE	Not null
Rank	VARCHAR(25)	Not null
Designator	VARCHAR(25)	Not null
Flight_Hours	INT	Not null

Type_Hours	INT	Not null
Dept_Head	BIT(1)	Default B '1'; not null
Status	VARCHAR(25)	Not null
Service	VARCHAR(25)	Not null
Aircraft_Type	VARCHAR(25)	Not null

Table 15: Answers Table Schema

Field	Type	Constraints
Survey_ID	INT	Primary Key, references Respondent, cascade deletes and updates
Squadron_ID	INT	Primary Key, references Respondent, cascade deletes and updates
Question_ID	INT	Primary Key, references Questions, cascade deletes and updates
Answer	INT	-1 >= Answer <= 6, not null

G. DATABASE DESIGNED FOR FLEXIBILITY

One of the major goals in the development of this software intelligent agent was to have all components dynamically created from reading the database, taking current version questions and creating the graphic user interface for the intelligent agent. The addition of the questionnaire table allows for this to be easily implemented. The tables were arranged so that each respondent can have answers associated with the version of the questionnaire they took. What this means is there is no longer a requirement to redesign the database if a question needs to be added, deleted, or modified as in the original database design.

H. OTHER DEVELOPMENT ISSUES

It is important to note that in developing this new database design, that adhering strictly to the relational model was a priority. While not always the best approach when developing a database, it was essential to the design of this database primarily for the

reason that this database would not have a full-time administrator to manage the database. As a result, the database will get very minimal maintenance, so the RDBMS must be able to protect the integrity of the database via business rules that are explicitly specified in the database design. The trade-off for this strict adherence to the relational model has been at the cost of more difficult query development and longer query running time, due to larger number of inner joins required when completing a query.

The database was designed using Microsoft Access 97 because of its ubiquitous presence, easy-to-use graphical interface, and compliance with the U.S. Navy's IT-21 policy. From that design, the database was populated with over 319 validated surveys from the original database to test the software intelligent agent. Once tested, the database design was defined in SQL-2 Data Definition Language (DDL), which is included in Appendix D. The Access database and the DDL were then provided to the commercial contractors who are building the production-level ACSA survey system for implementation in Microsoft SQL Server 7.0 within the NPS campus network domain.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERT SYSTEMS

A. INTRODUCTION

Dr. Edward Feigenbaum, Professor of Computer Science at Stanford University, describes an expert system as an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solutions. An expert system, according to Dr. Feigenbaum's description, should emulate the decision-making ability of a human expert. An expert is a person who has expertise in a certain domain, and, like the human expert, the expert system's expertise would be limited to a certain domain. The terms "expert system", "knowledge-based system", and "knowledge-based expert system" are often used interchangeably. An abstract conceptual model of the expert system is presented in Figure 5.

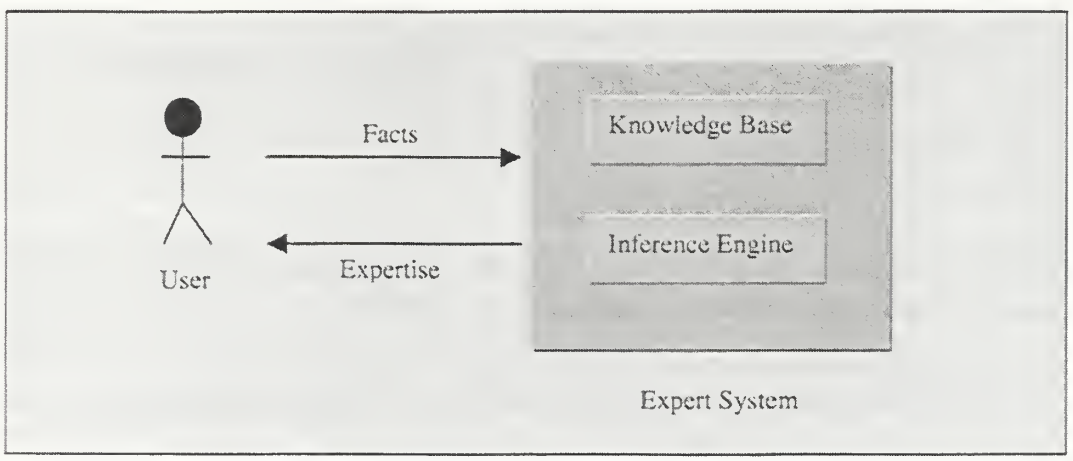

Figure 5: Expert System Conceptual Model

Expert systems are typically constructed from rules and facts, which are derived from domain knowledge and from expert knowledge. Rules model human problem solving, and facts are things that the system considers to be true. Rules and facts are collectively known as the knowledge base. An inference engine processes the rules and the facts to create new facts. An application that separates the knowledge base from the inference engine is known as a shell. The shell allows the knowledge base of a system to be changed as needed. Therefore, a unique inference engine does not need to be built for

every expert system. An expert system shell, like CLIPS or Jess, can be used in almost any expert system.

B. FUNDAMENTAL CONCEPTS

The following list briefly introduces the fundamental concepts of expert systems. The chapter discusses some of the concepts in more detail further below.

1. Experts

Experts are people who know how to solve complex problems within a particular domain. Experts know which facts are important. They understand the relationships between the facts and know how to make a plan to solve the problem. An expert typically exhibits the following characteristics:

- Recognizes and formulates the problem
- Solves the problem quickly and properly
- Can explain the problem and the solution
- Learns about the domain from experience
- Restructures knowledge whenever needed
- Knows when to break the rules
- Degrades gracefully by knowing the limits of their expertise, but still manages to develop a reasonable solution

An expert system would need to exhibit all of these characteristics to completely emulate the human expert. However, most expert systems exhibit only a subset of these characteristics. In general, most expert systems can solve a problem quickly and explain the solution. [Ref. 2]

2. Expert Knowledge

Expert knowledge is described as expertise in a specific domain that has been acquired over time from experience and training. Expertise includes the following types of knowledge:

- Theories

- Rules and procedures
- Heuristics
- Global strategies for solving problems
- Meta-knowledge
- Facts

Expert knowledge enables experts to make better and faster decisions than non-experts can when solving complex problems. [Ref. 2]

3. Knowledge Base

The knowledge base contains the facts and rules that the expert system will use to solve a problem. Facts and rules express the expert system's knowledge. The knowledge base is typically organized and used by the expert system via a process called knowledge representation.

4. Knowledge Representation

Knowledge representation schemes organize the knowledge contained in the knowledge base in a manner that enables the expert system to use the knowledge for reasoning. Thus, the knowledge base contains a data structure that can be manipulated by an inference engine. The inference engine searches and performs pattern-matching operations on the data structure. Knowledge representation schemes include:

- Propositional logic
- Predicate calculus
- Decision trees
- Semantic networks
- Production rules
- Frames

5. Knowledge Engineering

Knowledge engineering involves capturing expert knowledge and building a system that can use the knowledge. The process involves knowledge acquisition, validation, representation, and inferencing.

6. Inference Engine

The inference engine acts as the "brain" of the expert system. The inference engine knows how to use the system's knowledge base, and the engine organizes and controls how the system solves a problem.

C. KNOWLEDGE ENGINEERING

The facts and rules contained in the knowledge base represent many different types of knowledge. Types of knowledge include meta-knowledge, theories, facts, objects, relationships, heuristics, and general knowledge. All types of knowledge must be gathered from various sources, such as, human experts, databases, documents, observations, and books. Knowledge must then be validated and properly represented in the system. The knowledge engineering process involves acquisition, validation, representation, and inference. A knowledge engineering specialist, the knowledge engineer, conducts the knowledge acquisition process.

1. Acquisition

The knowledge engineer elicits knowledge from experts and extracts knowledge from books, files, and documents. Knowledge acquisition usually requires the cooperation of human experts. Eliciting knowledge from human experts is a complex and difficult task, because human experts often make decisions based upon "gut feelings" (perception, insight, and intuition) that have been developed from years of experience. These "gut feelings" are difficult to explain, and motivating the human expert to explain the reasons for making a particular judgement call requires a skillful interviewer. The knowledge engineer must have the ability to elicit and codify this type of subjective

undocumented expertise. Knowledge acquisition often creates a bottleneck in the development of expert systems [Ref. 3].

Elicitation of knowledge from an expert can be conducted manually or with automated computer support. Manual methods typically involve face-to-face interviews, which can be slow and expensive. Therefore, expert system development tends to involve some level of automated support.

Automated tools, such as, the Expertise Transfer System (ETS), provide the knowledge engineer with computer programs that interview experts and elicit knowledge via an interactive process. Automated systems can increase the productivity of the knowledge engineer by reducing the cost of knowledge acquisition. Automated systems use methods like data mining, rule induction, interactive induction, case-based reasoning, and neural computing to build a rule-based knowledge set.

2. Validation and Verification

Developers must validate and verify the knowledge base. Validation means that the knowledge base contains the right knowledge. Verification means that the knowledge base is constructed properly. The knowledge base can be verified by running test cases and testing the outcomes. Some knowledge acquisition programs possess automated verification capabilities.

3. Representation and Inference

Knowledge representation and reasoning/inferencing methodologies are discussed in the following two sections.

D. KNOWLEDGE REPRESENTATION

Knowledge representation involves storing and managing knowledge. Effective representation results in effective and competitive use of knowledge. Knowledge representation schemes can be programmed and stored in memory. Knowledge representation schemes are also constructed so that the knowledge contained in the data structure can be used in reasoning. The expert system reasons by manipulating the data structure to search the knowledge base and to apply pattern-matching techniques to make

inferences and conclusions about the knowledge base. Popular knowledge representation schemes include logic, semantic networks, production rules, and frames.

1. Logic

Knowledge may be represented by propositional logic and predicate logic. Generally, logical knowledge representation involves inputs (premises), logical processes, and conclusions (inferences). Logical processes produce conclusions based upon the inputs. Conclusions may then represent facts that are known to be true and may be used to make further inferences. Computers use symbolic logic to convert statements and the reasoning process into symbolic forms suitable for machine manipulation. Propositional logic (propositional calculus) and predicate logic (predicate calculus) are two fundamental forms of computational logic.

Propositional logic manipulates propositions by using symbolic logic. Propositional logic determines whether a statement (proposition) is true or false. The truthfulness of the statement can then be used for deriving new inferences or propositions. Propositional logic uses symbols to represent propositions, premises, and conclusions. For example, consider the following syllogism:

Premise P: All tigers are cats.
Premise Q: Harry is a tiger.
Conclusion R: Harry is a cat.

The meaning of the words is not important in propositional logic. Consider the same syllogism as above, except that all nouns have been replaced by symbols:

Premise P: All X are Y.
Premise Q: Z is an X.
Conclusion R: Z is a Y.

Propositional logic makes inferences based on form rather than on meaning. Therefore, propositional logic treats both syllogisms above as essentially the same syllogism. The concept of separating form from meaning makes propositional logic very powerful, because the validity of an argument can be considered objectively without the prejudice of semantics [Ref. 4]. Logic may also be considered as a mathematical theory in this regard and can be used to prove mathematical theorems.

All propositions may be represented with symbols and then manipulated using the rules of propositional logic to make inferences. The syllogism above may be further expressed with symbols in the following way: $P \wedge Q \rightarrow R$, which literally means, "If P is true and Q is true, then R is true." Symbols include logical operators like "$\wedge$" and "$\rightarrow$," which can connect two or more propositions to form propositions that are more complex. Table 16 depicts some common logical connectives.

Table 16: Logical Connectives

Connective	Meaning
$\wedge$	AND (conjunction)
$\vee$	OR (disjunction)
$\sim$	NOT (negation)
$\rightarrow$	if...then (condition)
$\leftrightarrow$	if and only if (bi-conditional)

Propositional logic is limited by the fact that propositional logic deals with complete statements only. Propositional logic cannot examine the internal structure of a statement; therefore, its ability to represent real-world knowledge is limited. In particular, propositional logic cannot prove the validity of propositions involving quantification like "all," "no," and "some."

Predicate logic, however, examines the internal structure of statements and can break a statement down into component parts. The component parts may be an object, a characteristic of the object, or some assertion about the object. The simplest form of predicate logic is called first order predicate logic, which is based upon first order theories. In first order theories, a set of objects is selected, and all the statements of the theory are statements about the objects. The set of objects is called the "domain," and the objects in the domain are called "individuals." Predicate logic uses quantifiers as described above to explicitly quantify other words to make statements more exact. Quantifiers permit a wider scope of expression than propositional logic permits. In fact, propositional logic is a subset of predicate logic.

A universally quantified statement has the same truth-value for all individuals in the same domain. The symbol $\forall$ represents the universal quantifier and is followed by one or more arguments for the domain variable. For example, in the domain of numbers

$\forall(x)$ $(x + x = 2x)$ means that for all x (where x is a number), the statement $x + x = 2x$ is true. [Ref. 4]

The statement "$x + x = 2x$" is a function and can be represented as a predicate function P(x). Therefore, the previous example can be written as $(\forall x)$ P(x). Predicate functions describe a property of the object.

An existentially quantified statement means that the statement is true for at least one member of the domain. The symbol $\exists$ represents the existential quantifier and is followed by one or more arguments for the domain variable. In the domain of numbers, the statement $(\exists x)$ $(x \bullet x = 1)$ means that there is some x whose product with itself equals one.

Artificial intelligence programming languages, particularly PROLOG, use predicate logic to represent knowledge. For example, let H be the predicate function for "human" and let M be the predicate function for "mortal." Then the statement that all humans are mortal can be written as $(\forall x)$ $(H(x) \rightarrow M(x))$, which means that for all x, if x is human, then x is mortal. Furthermore, the statement can be represented as a semantic net, as shown in Figure 6. The statement may also be expressed as a rule: "If x is human, then x is mortal."

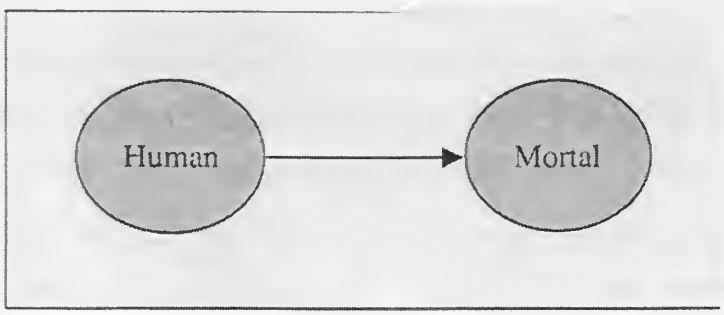

Figure 6: Semantic Net

2. Semantic Networks

Semantic networks graphically depict knowledge by showing the relationships between objects. In a semantic diagram, objects are called nodes and relationships are

called links. Figure 7 displays a semantic network, which is more complex than the network shown in Figure 6.

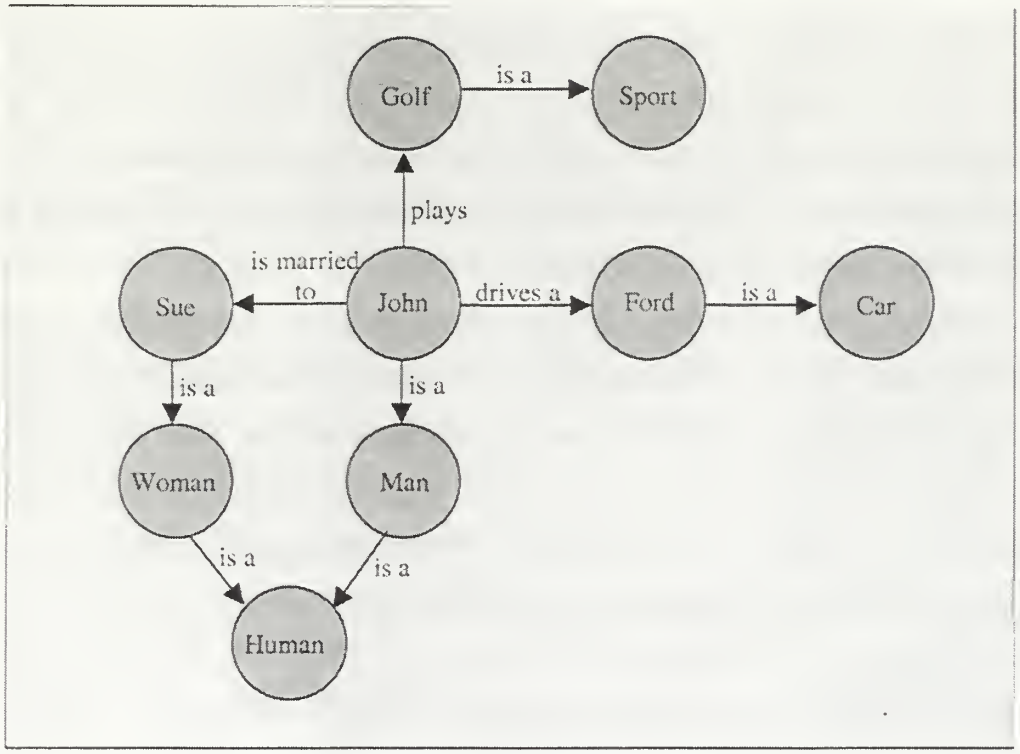

Figure 7: Semantic Network

Semantic networks provide a visual representation of relationships that can be combined with other knowledge representation methods.

3. Production Rules

Production rules represent knowledge via condition-action pairs, which typically look like IF-THEN statements:

- **Rule 1**: If an organizational safety area receives an assessment that is less than zero, then the organizational safety area is weak.

- **Rule 2**: If the survey contains three or more answers that are four standard deviations from the mean, then the survey may contain data entry errors.

Each rule in an expert system represents a small chunk of expertise. Although each rule can be considered independent of any other rule, the set of rules actually

becomes highly interdependent within an expert system. Changing one rule will often affect one or more rules in the knowledge base.

Rules may represent knowledge rules or inference rules. Knowledge rules are declarative and state all the facts and relationships about a problem. Rule 1 is an example of a knowledge rule. Inference rules are procedural and advise on how to solve a problem if certain facts are true. Rule 2 is an example of an inference rule. Inference rules can also define meta-rules, which are rules about rules. For example, a meta-rule could deactivate another rule if certain facts are true.

Representing knowledge with production rules is especially applicable if the expert system will need to recommend a course of action based upon facts. Rules simulate the cognitive behavior of the human expert, because rules express knowledge via a natural form of knowledge (IF-THEN). Production rules are easy to understand, maintain, and combine with uncertainty. However, complex knowledge may require thousands of rules, which can make rule management difficult.

4. Frames

Frames provide an object-oriented data structure for representing knowledge. A frame is a concise hierarchical structure that contains the characteristics and attributes for an object. An object may be a thing, a situation, or a problem. Expert systems rely upon frames extensively for describing domains and domain instances.

An example of a frame is as follows: person (name, hair_color, eye_color). The frame describes an object of the domain "person," and knowledge about the object is contained in "slots." The slots are name, hair_color, and eye_color. The frame could be used as follows: person (John, blonde, blue). Intuitively, the frame then describes an instance of person named John who has blonde hair and blue eyes.

E. REASONING METHODS

Expert systems solve problems by reasoning, which is also called inferencing. Ideally, humans reason and computers make inferences. Expert systems make inferences by applying logical processes on the knowledge base. Expert systems use rules, inference trees, frames, models, and cases to make inferences. Furthermore, inferences

may be made with certainty or uncertainty. First, this section discusses the five popular inferencing methods. Second, this section introduces the concept of reasoning under uncertainty.

1. Rule-based Inferencing

Rule-base inferencing uses either forward chaining or backward chaining to create a chain of inferences that connect a problem to a solution. In forward chaining, the chain proceeds from the problem to the solution. Therefore, forward chaining reasons from the facts to the conclusion. In backward chaining, the chain proceeds from the solution to the problem. The backward chain starts at the hypothesis and traverses back through the facts that support the hypothesis. If the chain is unable to find facts that support the hypothesis, then the hypothesis must be false.

Artificial intelligence programming languages, particularly PROLOG, use backward chaining for inferencing. Backward chaining is a goal-driven approach. Reasoning starts from a hypothesis, or a goal, and then seeks to verify the hypothesis as true or false. The inference engine attempts to find a rule that has the goal in its conclusion. For example, consider the following expert system that has the following information in its knowledge base:

- **Fact:** Dog (Rover)
- **Rule 1:** Dog (x) $\rightarrow$ Mammal (x)
- **Rule 2:** Mammal (x) $\rightarrow$ Animal (x)

Assume that the goal, or hypothesis, is Animal (Rover). In other words, the expert system will decide if Rover is an animal. Backward chaining starts with the hypothesis, Animal (Rover), and works backwards through an inference chain until it finds a fact that supports the hypothesis. In this case, the inference engine will follow the inference chain (Rule 1 and Rule 2) backwards to the fact, Dog (Rover). The inference engine will determine that the hypothesis, Animal (Rover), is true.

Expert system shells, particularly CLIPS, use forward chaining for inferencing. Forward chaining is a data-driven approach. Reasoning starts with all the known facts, or data, and then seeks to match the facts to the IF conditional elements of the IF-THEN

rules in the knowledge base. The inference engine makes conclusions as it tests each rule. If the facts match the conditional elements of the IF portion of the rule, then the conclusion must be true. For example, consider the same knowledge base as explained in the previous paragraph. The inference engine will examine each rule and look for matching facts. If matching facts are found, the engine will activate the rule and add new facts to the knowledge base if required. The new facts may subsequently cause other rules to be activated. In this case, the engine starts with Rule 1 and finds that Dog (Rover) matches the conditional element. The engine adds a new fact, Mammal (Rover), to the knowledge base. The engine then examines Rule 2 and finds that Mammal (Rover) matches the conditional element. The engine adds a new fact, Animal (Rover), to the knowledge base. Thus, the engine has successfully determined from the facts that Rover is an animal.

Forward chaining and backward chaining have characteristics that make them better suited for certain tasks. Forward chaining is well suited for problems involving planning, monitoring, making conclusions about the facts, and predictive analysis. Backward chaining is well suited for problems involving diagnosis, finding evidence to support the conclusion, and explanation. Although each chaining method has its own characteristics, that does not mean that one method cannot solve problems that the other method is better suited to solve. A forward chaining method can be used for diagnosis and a backward chaining method can be used for planning. Some expert system shells, like Jess, enable both forward and backward chaining.

2. Inference Trees

Inference trees depict the inference process schematically. In fact, some applications, like G2, can display the knowledge base visually with an inference tree. As shown in Figure 8, premises and conclusions are shown as nodes. Branches connect the premises to the conclusions. The connection between a premise and a conclusion represents a rule. Conclusions can be connected via AND and OR nodes. Inference trees do not provide deeper reasoning capabilities than other reasoning methods provide.

Inference trees simply provide a visual representation of the knowledge base, so that the structure of rules may be inspected and interpreted in a convenient manner.

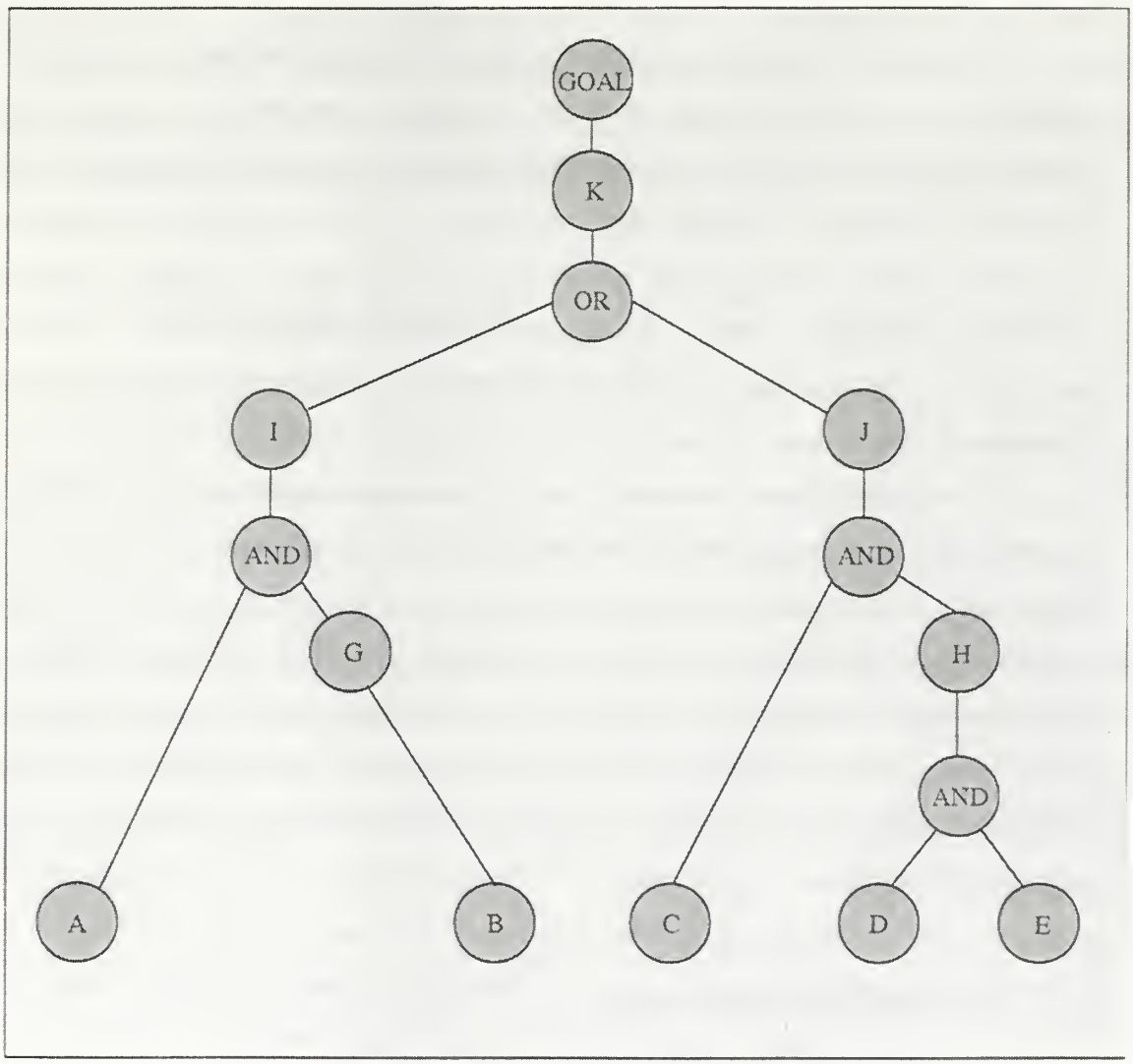

Figure 8: Backward Chaining Inference Tree

3. Frame-based Inferencing

Expert system shells commonly use frames and rules to make inferences about objects. A rule can reason about the characteristics of a frame by referring to the frame's slot value. For example, consider the following frame and rule:

- Frame: truck (weight)
- Rule: If the weight of the truck exceeds 2,000 pounds, then suggest a detour.

Then for the domain instance, truck (3,000 pounds), the expert system would suggest a detour.

4. Model-based Inferencing

Model-based reasoning depends on knowledge of the structure and behavior of a model, rather than depending on production rules that represent expertise. The knowledge engineer must create a complete and accurate model of the system in order to effectively implement model-based reasoning. Devices, such as machines and computers, possess behaviors that are relatively easy to model. Therefore, model-based reasoning is especially suitable for diagnosing difficult equipment failures. Model-based reasoning is often combined with other knowledge representation and inferencing methods like frames and rules.

Two types of general model types include mathematical models and component models. Mathematical models can describe behaviors via mathematics. A mathematical model may be used to describe the flow of water through a pipe. The model would account for the size of the pipe and the rate of flow to predict and diagnose the behavior of the system. A component model, however, would treat the pipe and the water as sub-components of a model. Each component would adjust its characteristics according to its behavioral model, and then each component would adjust itself via interaction with other relevant components.

5. Case-based Inferencing

Case-based reasoning creates new solutions based upon other solutions that were used to solve previous problems. This method requires that the system find relevant cases. Finding relevant cases involves [Ref. 3]:

- Characterizing the input problem by assigning appropriate features to it.
- Retrieving cases from memory that have the same features.
- Picking the case(s) that match the input best

Cases are the primary elements in case-based reasoning systems. A case may be as simple as a script that describes a well-known sequence of events. The case-based

inference engine finds a case that most likely matches the new problem by reviewing stored scripts. Then the script is applied to the new problem. Case-based reasoning is often combined with other knowledge representation and inferencing methods. If a domain cannot be formalized with rules, then case-based reasoning may offer a solution. The domain may not facilitate the formation of rules, because the domain has a weak causal model or contradictory rules may apply in different situations. Sometimes the output may be too complex to be expressed as rules, such as, battle plans or political analysis.

6. Reasoning With Uncertainty

Humans are very good at making decisions under uncertainty. Computers, however, are challenged when making certain decisions under uncertain conditions. The knowledge representation schemes and inferencing methods introduced so far have dealt with exact reasoning only. Exact conclusions have been deduced from exact facts. Real situations, however, often require conclusions to be made under uncertain situations. The goal of an expert system is to find the "best" solution, and if a system fails to deal with uncertainty, then the best solution may be overlooked. Unfortunately, determining the best conclusion is not always easy. Several methods have been developed for dealing with uncertainty, and the expert system designer must choose the appropriate method. Methods include various probability theories, certainty theories, and fuzzy logic.

All methods that deal with uncertainty attempt resolve errors that contribute to uncertainty. Different types of errors include ambiguity, incompleteness, incorrectness, randomness, measurement errors, systematic errors, and reasoning errors.

Probability theory provides a quantitative means for dealing with uncertainty. Pascal and Fermat first proposed classical probability in 1654. Classical probability is called a priori probability, because the theory deals with ideal systems without regard to the real world. An a priori probability assumes that all possible events are known and that each event is equally likely to happen. Formal probability theory is defined by three axioms:

- **Axiom 1:** $0 \leq P(E) \leq 1$
- **Axiom 2:** $\sum_i P(E_i) = 1$, *where all E_i are mutually exclusive*

- **Axiom 3:** $P(E_1 + E_2) = P(E_1) + P(E_2)$, *where E_1 and E_2 are mutually exclusive*

In each axiom, P(E) represents the probability of event E occurring. Axiom 1 defines the range of probability to be the real numbers from zero to one. If an event is certain, then the event's probability is one. Conversely, if an event is impossible, then the event's probability is zero. Axiom 2 declares that the sum of all mutually exclusive events is one. A corollary to Axiom 2 states that P(E) + P(E') = 1, where E' is the complement of E. Therefore, the occurrence and non-occurrence of an event are impossible. Axiom 3 states that if two events are mutually exclusive, then the probability of one or the other occurring is equal to the sum of their individual probabilities. The formal axiomatic definition of probability theory supports the deduction of probability theories outside of a priori probability. For example, experimental probability (a posteriori probability) describes P(E) as the limit of a frequency distribution: $P(E) = \lim_{N \to \infty} (f(E) / N)$, where f(E) is the frequency of outcomes of an event for N observed total outcomes. Experimental probability induces an event's probability from an event's frequency in a large number of trials.

Subjective probability is not supported, however, by axioms or empirical measurements. Instead, subjective probability assigns probabilities based on beliefs, experience, judgement, and opinions. In fact, uncertainty is represented as a degree of belief called a certainty factor, rather than as a probability. Types of subjective probability theory include certainty theories and fuzzy logic. Subjective probability allows expert systems to reason about non-repeatable events like medical diagnoses and mineral exploration. Realistically, any medical diagnosis is a conditional probability that can be interpreted as a degree of belief that the conclusion is true. All patients are unique, so the problem is non-repeatable. Therefore, a medical diagnosis does not represent a probability, but rather a degree of belief that the diagnosis is correct.

F. THE JAVA EXPERT SYSTEM SHELL (JESS)

The Java Expert System Shell (Jess) is an expert system shell written completely in the Java programming language. Jess supports the development of rule-based expert

systems. The expert system can be written entirely with scripts or Java, or partially with scripts and partially in Java. Jess was written by Ernest J. Friedman-Hill at Sandia National Laboratories, and it can be downloaded at no cost on the World Wide Web at http://herzberg.ca.sandia.gov/jess.

Jess was originally a porting of the CLIPS expert system shell, which is written in C, to the Java language. Jess has diverged from CLIPS during its continuous development, but Jess retains many similarities to CLIPS. A person familiar with CLIPS and Java should have little difficulty in learning how to use Jess.

Jess uses a very efficient algorithm for matching facts with the conditional elements of rules to determine which rules have had their conditions satisfied. The algorithm is called the Rete Matching Algorithm. The Rete Algorithm is necessary for efficiency, because the inference engine must constantly examine the facts for matches. If the engine needed to examine the fact list only once, then no algorithm would be necessary. However, the matching process occurs repeatedly in rule-based expert systems. Therefore, an efficient method for pattern matching must be used. The alternative is for the inference engine to inspect every rule and every fact during each execution cycle.

The Rete Algorithm takes advantage of the temporal redundancy exhibited by rule-based expert systems. Temporal redundancy refers to the idea that facts change slowly over time. Only a small number of facts are added or removed between each execution cycle. Therefore, the engine does not need to examine every fact during each cycle. The Rete Algorithm achieves its efficiency by remembering previous matches between cycles and by knowing which facts have changed. During an execution cycle, the engine determines only how the new facts have affected the state of the matching process.

Jess contains the Java classes necessary for creating a knowledge base and an inference engine. Creating an expert system in Jess involves writing rules in Jess syntax, loading facts into the knowledge base, and then running the inference engine. Both rules and facts may be loaded into Jess via scripts that are written in the Jess syntax. The Jess distribution includes a parser class that will parse a script and load it into the knowledge

base. Facts may also be loaded from Java via the Rete.assert() method, the Funcall.add() method, or the Rete.executeCommand() method.

JavaBeans may also be added to the knowledge base as facts. The advantage to this Jess feature is that the slot values of the fact will always match the attributes of the bean. Jess uses the java.beans.PropertyChangeSupport class and reflection to update facts in the knowledge base that represent beans in a Java application. The ACSA Safety Agent application takes advantage of this feature by using JavaBeans for Question objects and for Safety Area objects.

G. THE ACSA SAFETY AGENT AND JESS

1. Installing Jess

Download Jess from http://herzberg.ca.sandia.gov/jess. The distribution should include specific installation instructions, but the procedures for installing Jess 5.0 is briefly covered here. Unzip the archive. Place the Jess package in the machine's classpath environment variable. Compile the Jess package. If the package compiles successfully, then Jess is installed and is ready for use.

2. Writing Jess scripts

The ACSA Safety Agent loads fact templates and rules into the the knowledge from a script file. The script, rules.clp, is included in Appendix C. Jess scripts contain code written in Jess syntax and can be parsed by the Jess Parser class. Jess syntax is very similar to CLIPS syntax and is well documented in the distribution.

The rules.clp file is located in the safetyAgent.agentApp.resources package. The location allows the application to use the Class.getResource() method to load the file, so that the file may be loaded regardless of the directory from which the application is run. To change a rule in the ASCA Safety Agent application: open the script in a text editor, modify the contents, and save the script in the safetyAgent/agentApp/resources directory. If the file is moved from this directory, the agent will be unable to find the rule file.

3. Using Jess with Java

Jess may be used with Java in many different ways. The difficult decision is deciding which method is appropriate. Rules are best implemented via scripts, since the rules tend to be static. Facts, however, can be implemented by loading scripts, calling methods, or via JavaBeans. The ACSA Safety Agent calls methods and uses JavaBeans for managing facts in the knowledge base. A fact may be loaded via a script if the fact is known to be true every time the script is loaded. Loading facts via method calls are useful for facts that need to be added at certain points during execution. Using JavaBeans for facts are useful when the attributes of the fact will change frequently.

The ACSA Safety Agent uses beans for the Question and Safety Area facts, because the statistical data about the objects may change frequently and questions and safety areas are tied to the user's preferences. A user may de-select a question or a safety area from the assessment process at any time. Facts that are implemented as JavaBeans are updated dynamically as the attributes of the beans change. This feature makes coding much easier, because the code does not need to make method calls to retract, modify, and re-assert a fact every time a slot's value changes.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SOFTWARE AGENTS

A. INTRODUCTION

Software agents act as computerized servants. Humans create and employ agents, but humans typically neither directly supervise nor direct an agent's activities. Humans define an agent's capabilities and behaviors, and the agent acts in a relatively autonomous manner in accordance with their programmed capabilities and behaviors. Agents can sometimes learn and adapt their behavior accordingly. Agents can usually communicate, cooperate, negotiate, and even collaborate with other agents. They can assist people and can ideally interact with people in human-like ways. Some software agents attempt to simplify the use of computers by improving human-computer interaction. For example, word processors and spreadsheet programs, like Microsoft Word and Excel, use agents as "wizards" to assist users and to automate complex tasks. Microsoft Office uses agent technology in its animated assistants like the paperclip that taps on the monitor when it has an idea. This chapter discusses how software agents are defined, the characteristics of agents, the reasons for choosing an agent, the difficulties of producing intelligent software agents, and current agent technologies. This chapter will also discuss aspects of the ACSA intelligent agent where appropriate; however, a more detailed discussion is provided in Chapter VI, "Development of the ACSA Intelligent Agent."

B. WHAT ARE SOFTWARE AGENTS?

Software agents are programs, which carry out tasks by proxy on behalf of a user or a group of users. Agents are often called software robots, softbots, or bots. Agent capabilities vary widely from basic skills, such as, automating straightforward routine tasks, to advanced skills, such as, adapting to user routines and preferences and negotiating on someone's behalf. The characteristic that distinguishes agents from other software programs is that agents can automatically adapt their behavior to the conditions they encounter and make simple to complex decisions without specific on-the-spot instruction from the user. An agent's degree of autonomy depends upon the agent's level

of sophistication. An agent's capacity for autonomous behavior may include taking the initiative, such as, providing the user with information that is not specifically requested by the user but may be of interest to the user. Software agents can be described in terms of agency, intelligence, and mobility. [Ref. 5]

1. Agency

Agency is the degree of autonomy and authority that is entrusted to an agent. Autonomy can be measured qualitatively by the nature of the interaction between the agent and the other entities in the system. The degree of agency is enhanced if an agent represents a user and acts as an agent for a user. Agency is an essential characteristic of an agent. An advanced agent can interact with other entities such as data, applications, or services. More advanced agents can collaborate and negotiate with other agents. [Ref. 5]

2. Intelligence

What exactly makes an agent "intelligent" is difficult to define, and the definition of intelligence has been the subject of many discussions in the field of artificial intelligence. A definitive answer has yet to be found. This chapter, however, will use the following definition as a working definition:

> Intelligence is the degree of reasoning and learned behavior: the agent's ability to accept the user's statement of goals and carry out the task delegated to it.
>
> At a minimum, there can be some statement of preferences, perhaps in the form of rules (an Expert or Knowledge Based System), with an inference engine or some other reasoning mechanism to act on these preferences.
>
> Higher levels of intelligence include a user model or some other form of understanding and reasoning about what a user wants done, and planning the means to achieve this goal.
>
> Further out on the intelligence scale are systems that learn and adapt to their environment, both in terms of the user's objectives, and in terms of the resources available to the agent. Such a system might, like a human assistant, discover new relationships, connections, or concepts independently from the human user, and exploit these in anticipating and satisfying user needs. [Ref. 6]

3. Mobility

Mobility is the degree, to which, agents can travel through a network. Agents may be static or mobile. A static agent may reside on a client machine (to manage a user interface, for instance) or reside on the server. Mobile agents may be created on one machine and then shipped to another for execution in a suitably secure environment. In this case, the program travels before execution, so no state data is included with the program. However, agents may also be mobile with state and transported from machine to machine in the middle of execution carrying accumulated state data with them. Such agents may be viewed as mobile objects, which travel to agencies where they must present their credentials to obtain access to services and data managed by the agencies. Agencies may also serve as brokers or matchmakers, bringing together agents with similar interests and compatible goals, and providing a "meeting point" where agents can interact safely. [Ref. 5]

C. WHY USE AN INTELLIGENT AGENT?

Computers are increasingly connected to each other and to larger networks, including the Internet. Data and information flow at the speed of light between millions of computer systems day and night seven days a week. The potential for information overload is tremendous. Too much data arriving at inhuman speeds and amounts can make it difficult to find information that is useful. Therefore, there is a growing need for tools to help manage information collection, management, and analysis. Intelligent agents, which are sophisticated software tools that incorporate aspects of artificial intelligence, act independently on the user's behalf, and may offer solutions to some major computer and database management needs.

Information access and management is an area of great activity, given the rise in popularity of the Internet and the explosion of data warehouses available to users. In the past few months alone, the on-line ASCA database has grown more than 40 percent in size from 250 respondents to over 350 respondents. More importantly, though, NPS has

continued to issue survey numbers to squadrons, and the size of the database is projected to grow at a higher rate than before. As the database grows in size, the amount of data that researchers can analyze will become a smaller percentage of all data in the database. Therefore, researchers need a tool that can analyze the database for them automatically and discover anomalies and make conclusions based on rules that express the researchers' expert domain knowledge. For example, the ACSA Safety Agent can inspect the ACSA database for data integrity and can make decisions about organizational safety assessment information contained in the database. The ACSA Safety Agent can generate and deliver safety assessment and database integrity reports automatically to a variety of destinations, including email, printer, file, or computer display.

D. DIFFICULTIES IN BUILDING AN INTELLIGENT AGENT

Agents and agent technology have been an active area of research in the artificial intelligence and computer science community for a number of years. Many universities have developed intelligent software agents. A number of companies deliver software agents capable of performing a wide variety of specialized tasks. However, each of these agents were handcrafted for a particular application. Building an intelligent software agent is a difficult and time-consuming task that requires an understanding of advanced technologies, such as knowledge representation, reasoning, network communications methods and protocols, and distributed programming. Sophisticated applications often require expertise in machine learning and machine planning technology.

A developer using intelligent agents in a new application must decide on the overall agent processing architecture, the agent's reasoning mechanism and associated pattern matching technology, internal knowledge and data representations, agent-to-agent communications protocols, and message formats. In addition, if the agent is to learn from its environment or its owner, then some kind of machine learning technology will be required. Sophisticated agents may require a planning capability, and this will require that the developer select a planning algorithm and implementation. If the application requires multiple communicating agents, then the developer must establish a robust

communications protocol between the agents. Therefore, the developer will need to know about the underlying communications technologies used for inter-agent communications.

E. AGENTS IN USE TODAY

Intelligent agents are currently being used or developed for a variety of uses, such as, personal assistants, information gathering, data mining, and telecommunications. Since these software agents perform specific tasks as directed by the user, these programs are often called *bots*, a nickname that quickly identifies them as on-line intelligent agents that behave like software robots. BotSpot, a website resource for bot information, lists over 1,000 bots on its website at http://bots.internet.com. The site lists the bots by category, of which, there are eighteen. Categories include commerce bots, email bots, data mining bots, chat bots, game bots, government bots, fun bots, news bots, newsgroup bots, search bots, shopping bots, stock bots, software bots, knowledge bots, and update bots. This section, however, limits its discussion of example intelligent agent software programs to the following more general categories of personal applications, business applications, telecommunications, expert systems, computer diagnostics, network management and monitoring, and email filtering and sorting.

1. Personal Applications

Personal applications include agents that perform services for an individual person, such as, gathering news stories or helping a user write a letter in Microsoft Word. Two examples of personal application agents include Paperboy and Microsoft Office Assistants.

Paperboy, which is offered by Heurics Systemhaus, compiles a personal newspaper from daily newspapers based on user-specified topics and delivers a personal daily newspaper via email to the user. The agent uses a rule-based search engine that searches U.S. and European publications for newspaper stories that may satisfy the user's preferences. Online searching via keyword or complex queries is also possible and very fast. Paperboy is available on the World Wide Web in English and in German at www.paperboy.de.

Microsoft includes agent technology in all of its Microsoft Office products. The most obvious agent in Microsoft Office is the Office Assistant, which appears as an animated paperclip by default. The agent can also be configured to appear as an exclamation point, Albert Einstein, a smiley face, a red rubber ball, or some other assistant that is offered in Microsoft Office. More assistants can also be download from the web. Microsoft has attempted to make the assistants' look-and-feel as personally configurable as possible. Office Assistants can help users with many tasks in Microsoft Office, such as, formatting a letter or creating a table. The agent will also appear if the agent determines that the user might be doing something where the agent might be able to help. For example, if a user starts to write a letter, the agent may automatically appear as soon as the user enters "Dear Mom,". The agent will tell the user that it appears that the user is writing a letter and will ask the user if he or she would like help with the format of the letter.

Financial planning services use agent-driven expert systems to reduce the costs of their services. Agents have made the planning process quicker, easier, and more consistent. Financial plans can help individuals with insurance, retirement, investments, income taxes, estate planning, and cash and debt management. The plans have general rules and should be flexible enough to accommodate personalized options, risk attitudes and individual preferences.

2. Business Applications

An intelligent agent in use at Hewlett-Packard Co. is currently helping automate a quarterly wage-review process that covers approximately 13,000 salespeople. The software, which runs on a personal computer, performs essentially the same tasks as a team of 20 administrators. [Ref. 7]

The implementation of a knowledge-based system at John Hancock Mutual Life Insurance Company provides an example of the potential for incorporating agents into core business processes. John Hancock processes a very large volume of new insurance applications, in excess of several hundred thousand applications each year. New life insurance applications come from over 300 field offices maintained by John Hancock across the country. Underwriting had always been performed at corporate headquarters.

In 1986, an internal technology consultant in the corporate data processing department considered a variety of new technologies as they might be applied to insurance. The consultant then developed a plan for applying expert systems. With the participation of five experienced underwriters and twelve computer programmers, the consultant completed the system in four years. Currently, the expert system does a large portion of the work that had to be done manually before. The applications from over 300 field offices are now processed through the expert system. [Ref. 7]

The very nature of the John Hancock's business dictated a "front-end" focus for initial expert systems development efforts, because the requirement was to automatically approve as many new cases as possible without human intervention. The nature of John Hancock's retail life insurance business, i.e., the large volumes of new applications, was highly amenable to a production-oriented underwriting system. [Ref. 7]

The banking industry has used expert systems in many cases with good results in areas concerning trade decisions, credit decisions, loan applications, and opening new accounts. As technology makes banking convenient, customers can access banking services and do banking transactions any time and from anywhere.

Many financial companies use expert systems for credit analysis. The benefits of using expert systems for credit analysis are speed and accuracy, both of which far exceed human capacity. American Express uses expert systems to process unusual requests. The system, called Authorizer's Assistant, can process requests much quicker than a customer service representative can, and the agent has decreased errors by nine percent.

3. Telecommunications

Telescript is part of an environment designed to serve as a platform for communications applications (including intelligent agents) and a foundation for personal communicators. Matsushita, Motorola, Philips, and Sony (members of the General Magic alliance) are licensees of the new technology and are building hand-held Magic Cap communicators.

The Magic Cap environment includes most modes of communication currently available, including:

- fax

- public electronic mail services
- telephones

Telescript technology allows users to launch their intelligent agents to perform tasks such as screening, routing, and delivering electronic correspondence. They will also be able to shop for goods and services and seek out time-critical information.

AT&T's PersonaLink Services, which uses the Telescript technology, will support smart messaging and host a variety of service providers, from personal newspapers to travel services to electronic shopping.

Integrated with its core communication capabilities, Magic Cap software includes features to help manage personal information: address cards that automatically get updated as the sender's information changes, a calendar that issues invitations to meetings, and a notebook that supports free-form and structured notes.

Magic Cap software will be available in handheld communicators and as software running on personal computers, so customers can use it whether they are mobile or at their desks. In addition, for those who use the software on a personal computer and also own a Magic Cap communicator, appointments, addresses, and other data will be reconciled when the two are connected.

Since both the Magic Cap hardware standard and Magic Cap software are extensible, manufacturers will be able to create a variety of Magic Cap-based products that incorporate their unique expertise and differentiate their products from others in the market. Two-way communication is built into every Magic Cap communicator. Customers will have a range of Magic Cap products to choose from-wireless and wireline, mobile and stationary -- all of which will take advantage of the growing library of products and services developed by independent vendors and all of which will communicate.

4. Expert systems (specialized knowledge-based applications)

One of the results of research in the area of artificial intelligence has been the development of techniques that allow for the modeling of information at higher levels of abstraction. These techniques are embodied in languages or tools that allow programs to

be built that closely resemble human logic in their implementation and are therefore easier to develop and maintain. These programs, which emulate human expertise in well-defined problem domains, are called expert systems. The availability of expert system tools, such as CLIPS and Jess, has greatly reduced the effort and cost involved in developing an expert system.

Rule-based programming is one of the most commonly used techniques for developing expert systems. In this programming paradigm, rules are used to represent heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. Some of the uses of expert systems today include, but are not limited to:

- Medical use (diagnosis processing, treatment regimens, patient record/payment info-tracking, and insurance coverage processing)
- Law (case classification and analysis, precedent tracking)
- Law enforcement (suspect identification, case comparison, and info-tracking)
- Translation services (conversion of text written in one language into another language)
- Customer trouble-shooting services (help desk services, repair requests)

5. Computer Diagnostics

Intelligent agents are often used to do check-ups on a computer's operating system. Sometimes intelligent agent software can configure the computer (installing programs, setting preferences, etc.) as well.

If a computer is a stand-alone unit, the intelligent agent software can be loaded onto the computer via a floppy disc or CD-ROM. If the computer is part of a network, then someone authorized to do computer or network management can use intelligent agent software to access the operating system of each of the computers on the network to do the following kinds of tasks:

- Diagnose problems on a specific computer
- Re-configure or make adjustments to the operating system
- Install and uninstall programs, documents, or miscellaneous stored data. In business, this process ensures that individual desktop computers have

sufficient storage space by routinely archiving documents of a particular type that were created either by a someone who has since left the company or before a certain date.

These types of applications are becoming more and more important as ways that computer equipment can be made ready for use by the first user, the next user, and so on. These applications also make it easier for re-sellers or re-users to identify and address problems on individual computers.

6. Network Systems Management and Monitoring

Network administrators use intelligent agent software to observe operations in progress, monitor use of particular resources or programs, and provide fairly complex use reports. These functions allow the network administrators to decide, for example, when the network is likely to need more storage capacity or when it may be necessary to do archiving of old data to free up space on the hard drives of various individual computers.

These types of tasks are now often handled from the network server level rather than having service personnel go around and access each computer individually. This can make an enormous difference in the time it takes, for example, to upgrade versions of software used by each of the computers in a company.

In addition, this type of network use of intelligent agent software allows commercial software vendors to upgrade or trouble-shoot the software directly, over the Internet and then via the company's network server.

Computer networks are increasingly critical to an efficient and effective work environment. Unfortunately, there are many things that can adversely affect an optimally functioning network system, such as

- power surges or power interruptions
- computer viruses
- unauthorized use of the system (system break-in from outside or damage caused by internal users)

Networked systems allow users access to many useful resources. Unfortunately, the connected nature of networked systems means that the system may be adversely

affected through its connections. Whether the damage is intentional or unintentional, intelligent agent software can help find out what went wrong and help get the system up and running again much more quickly than was previously possible.

7. E-mail Filtering and Sorting

E-mail has quickly become one of the single most important business tools in use today. It allows a much greater and more efficient system of communication for many business users. However, e-mail is subject to the same types of problems that the previous paper mail system has had problems with:

- junk e-mail, "spam," or otherwise unsolicited e-mail
- unwanted e-mail that was sent by people you know, but that you don't want to reply to immediately
- mass mailing e-mail and misdirected e-mail

Intelligent agent software offers some fairly sophisticated techniques for handling e-mail, such as:

- sorting
- filtering
- routing
- prioritizing

In addition to these duties, more complex e-mail intelligent agents can *learn* your preferences. For example, the intelligent agent may recognize an address that the user always ignores messages from. The intelligent agent may -- even if the user never instructed the agent to do this specifically -- learn that you don't consider messages from that source important and, over time, may assign these messages lower and lower priority.

F. AGENT TECHNOLOGIES

There is a tremendous amount of advancing technologies that will rely heavily and contribute to future agent development. Most of these technologies can fit into one of three categories discussed below:

1. Programming Languages

The use of object-oriented programming technologies, including object-oriented operating systems, databases and programming techniques, will make it easier to develop intelligent agents that are able to search across a variety of computer hardware and software platforms. Java seems to be the language of choice for building agents, because Java is object-oriented and has robust built-in libraries for networked programming. Software developers can create powerful intelligent agents in Java that execute on a wide variety of computer platforms and operating systems.

2. Intelligent Reasoning

Intelligent reasoning is the true backbone of agent technology and development. The quest to develop even smarter intelligent agents is an ongoing process that will include many cutting edge technologies. Some of the main technologies supporting intelligent agents are:

- Neural network technology is excellent at identifying relationships and automatically learning to recognize patterns based on *digestion* of large amounts of data. The ability to learn is an important characteristic for intelligent agents. Neural network technology expects to be closely linked to the development of intelligent agents in the future.

- Fuzzy logic enables software and hardware to handle imprecise data will make it very important in handling tasks where it is difficult to precisely define information that is sought or tasks that must be completed.

- Genetic algorithms help in automating the learning process for intelligent agents. This technology causes computer programs to mutate, with the result that they evolve into a series of new programs. The value of these mutant programs is then evaluated externally. The most valuable mutant programs are selected to mutate again. Using this Darwinian process of selection, optimum programs for a given environment are generated relatively quickly. Integrating

genetic algorithms with fuzzy logic will allow fuzzy logic programs to optimize themselves or to adapt to changing conditions.

- Expert Systems capture knowledge from a human expert and encode the logic into a computer system. This approach requires intensive human intervention in capturing and encoding knowledge. Expert systems consisting of a relatively small number of rules may be embedded into intelligent agents, but this technology is not likely to be used extensively as expert systems perform very poorly when data is imprecise.

3. Communications

In order for an intelligent agent to be useful, it needs to have the ability to communicate efficiently and with ease. While there are many technologies expanding the way the world looks at everything, a couple of communications technologies are very promising:

- Computer systems that can understand written, typed or spoken language will be important for intelligent agents to be successful. The use of natural language to tell an agent what to do will make the agent considerably more useful to computer users.

- Increased computing speed and densities will make it possible for larger, more sophisticated intelligent agents to run in the background without hindering other tasks that a computer needs to do. This includes higher speed modems, DSL, and cable

G. AGENT SOFTWARE PACKAGES AND VENDORS

There are many commercial vendors and research projects that support software intelligent agent development. Software agents and agent systems are modeled object-oriented using the latest findings of software architecture. This approach describes the software architecture as a system of components whose interactions are realized via connectors. With the architecture-based methodology, theoretical fundamentals for the implementation of computer-independent platforms for agent programming have been found. While this is in no way an exclusive list of all agent software and vendor's packages, the following is a listing of some of these packages and a brief description of how they work [Ref. 8]:

1. AgentBuilder by Reticular Systems, Inc.

AgentBuilder is an integrated tool suite for constructing intelligent software agents. AgentBuilder consists of two major components - the Toolkit and the run-time System. The AgentBuilder Toolkit includes tools for managing the agent-based software development process, analyzing the domain of agent operations, designing and developing networks of communicating agents, defining behaviors of individual agents, and debugging and testing agent software. The run-time System includes an agent engine that provides an environment for execution of agent software.Agents constructed using AgentBuilder communicate using the Knowledge Query and Manipulation Language (KQML) and support the performatives defined for KQML. In addition, AgentBuilder allows the developer to define new interagent communications commands that suit his particular needs.

All components of both the AgentBuilder Toolkit and the Run-Time System are implemented in Java. This means that agent development can be accomplished on any machine or operating system that supports Java and has a Java development environment. Likewise, the agents created with the AgentBuilder Toolkit are Java programs so they can be executed on any Java virtual machine. The AgentBuilder toolkit is designed to provide the agent software developer with an integrated environment for quickly and easily constructing intelligent agents and agent-based software.

2. Agentx by International Knowledge Systems

Agentx is a second generation, state of the art, set of lightweight, high performance, and scaleable distributed computing libraries for the Java programming environment. The libraries were designed to provide object request broker facilities that were easier to use, faster, more compact and highly functional, than the RMI libraries bundled with the Sun JDK or generally available Java based CORBA implementations. Unlike the aforementioned tools Agentx does not require the use of an interface definition language nor force the creation of stub or skeleton code. In addition Agentx is fully compatible with the Sun, IBM, and Microsoft virtual machines as well as the latest versions of both Internet Explorer and Netscape Navigator.

In addition to ORB functionality, Agentx also provides programming support for the creation and release of autonomous mobile agents. Objects can move around on the network, attach themselves to a host, and begin independent execution of their program code. This new technology allows for a complex computer program, or computationally intensive task, to be broken down into smaller discreet parts and distributed across a network of heterogeneous machines for simultaneous but independent execution. The computing devices performing the processing tasks might include desktop PCs and workstations, as well Java based network computers, or machines running a Java OS. Using this new type of software architecture, Agentx can potentially simulate the functionality of a large mainframe or supercomputer, for only a fraction of the cost.

Agentx, is targeted for sale to information systems groups within military, government, educational and corporate organizations, as well as to independent software companies for integration into their own products.

3. Microsoft Agent by Microsoft Corporation

Microsoft Agent is a set of programmable software services that supports the presentation of interactive animated characters within the Microsoft Windows interface. Developers can use characters as interactive assistants to introduce, guide, entertain, or otherwise enhance their Web pages or applications in addition to the conventional use of windows, menus, and controls.

Microsoft Agent enables software developers and Web authors to incorporate a new form of user interaction, known as conversational interfaces, that leverages natural aspects of human social communication. In addition to mouse and keyboard input, Microsoft Agent includes optional support for speech recognition so applications can respond to voice commands. Characters can respond using synthesized speech, recorded audio, or text in a cartoon word balloon.

The conversational interface approach facilitated by the Microsoft Agent services does not replace conventional graphical user interface (GUI) design. Instead, character interaction can be easily blended with the conventional interface components such as windows, menus, and controls to extend and enhance your application's interface.

Microsoft Agent's programming interfaces make it easy to animate a character to respond to user input. Animated characters appear in their own window, providing maximum flexibility for where they can be displayed on the screen. Microsoft Agent includes an ActiveX control that makes its services accessible to programming languages that support ActiveX, including Web scripting languages such as VisualBasic Scripting Edition (VBScript). This means that character interaction can be programmed from HTML pages.

4. Bee-gent by Toshiba Corporation

Bee-gent is a new type of software development framework in that it is a 100% pure agent system. As opposed to other systems that make only some use of agents, Bee-gent completely "agent-ifies" the communication that takes place between software applications. The applications become agents, and agents carry all messages. Thus, Bee-gent allows developers to build flexible open distributed systems that make optimal use of existing applications.

The Bee-gent framework is comprised of two types of agents. "Agent wrappers" are used to agentify existing applications, while "mediation agents" support inter-application coordination by handling all communications. The mediation agents move from the site of an application to another where they interact with the agent wrappers. The agent wrappers themselves manage the states of the applications they are wrapped around, invoking them when necessary. Thus inter-application coordination is handled through the agent wrappers generating and receiving requests, which are transported around by the mediation agents. The mediation agents do more than just transport the messages; they are able to respond to the nature of the request to determine the best course of action.

5. Cable by Logica Corporation

Cable is a generic system architecture developed by Logica as part of the GRACE Consortium. Cable can be used to develop and execute distributed applications that are based on the metaphor of multiple, cooperating intelligent agents.

Cable provides the user with an Agent Definition Language (ADL), for defining agents, and a parser known as the Scribe, for compiling agent definitions written in ADL into agent applications. Agents are developed using ADL and C++. ADL allows developers to use Cable without worrying about underlying detail, providing a language with a level of abstraction close to that of the agents with which an application is designed. Inter-agent communication over a local area network is handled using ORBIX, an implementation of the CORBA 2.0 standard.

6. JATLite developed at Stanford University

JATLite is a set of Java packages that make it easy to build multi-agent systems using Java. JATLite provides a basic infrastructure in which agents register with an Agent Message Router facilitator using a name and password, connect/disconnect from the Internet, send and receive messages, transfer files, and invoke other programs or actions on the various computers where they are running. JATLite facilitates construction of agents that send and receive messages using the emerging standard communications language, KQML. The communications are built on open Internet standards, TCP/IP, SMTP, and FTP.

7. JATLiteBean developed at University of Otago

JATLiteBean takes the KQML-speaking functionality of JATLite and wraps it into a JavaBean. JATLiteBean provides an easier-to-use interface to JATLite features including KQML message parsing, receiving and sending. It provides an extensible architecture for message handling and agent "thread of control" management. It also supports automatic advertising of agent capabilities to facilitator agents. Also included is a generic configuration file parser and KQML syntax checker.

8. Process Link developed at Stanford University

The Process Link project is developing an agent-based framework consisting of generic agents and a message protocol for integrating multi-disciplinary engineering software and for managing distributed design projects. The framework allows developers

to wrap legacy software with back-end code that will disturb the existing software interface as little as possible while providing useful coordination functions.

Researchers are using a "weak" agent technology, in which the wrapped software become agents that send messages corresponding to interaction semantics. The software is not necessarily "smart" and does not conform to any particular theory of agent construction.

VI. DEVELOPMENT OF THE ACSA SAFETY AGENT

A. INTRODUCTION

Technology exists today that can be used to build software that is intelligent, mobile, able to learn, and that can assist humans in dealing with the challenges of information overflow. Software can perform tasks that seem inhuman. In seconds, the ACSA Safety Agent can examine over 19,000 survey answers and determine which answers are four standard deviations outside the average for all answers. A year from now, the agent may be able to complete the same task in microseconds. The challenge, however, is making software that can perform tasks that seem human. Imagine a human being that can think as fast as a computer, or imagine a computer that can reason like a human being. Imagine not being able to distinguish between the human and the computer. Every step made in artificial intelligence, expert systems, information systems, processing hardware, and computer programming research takes mankind closer to turning such imaginations into reality.

The primary goal of this thesis was to use the most current technology available to build an assistant for conducting organizational safety assessment analysis. All of the major software components of this thesis required beta software, which is unsupported and undocumented. Beta software reflects either the cutting edge of technology or the next generation of a software product. This thesis brings together the latest in Java, networks, artificial intelligence, expert systems, software design, and database theory to create two significant contributions to the fields of aviation safety and computer science. The first contribution is a model for developing intelligent agents. This contribution is perhaps the most important contribution, because the model may be used for future development that will further advance the field of intelligent agent technology. The second contribution is a working application, the ACSA Safety Agent, which demonstrates the potential of the model. The ACSA Safety Agent application also contributes to the ACSA's purpose, which is to improve the safety process at the local command level and to encourage overall process improvements in all naval aviation units.

B. SOFTWARE DEVELOPMENT PROCESS

The design and implementation of the ACSA Safety Agent model and application followed a relatively simple software development process. The process involved requirements analysis, design specification, and implementation. Requirements analysis is the process of determining and documenting the user's needs and constraints, and the requirements analysis can be viewed as the design of a set of goals for the proposed system [Ref. 9]. Design Specification includes the specification of system functions, architecture, and interfaces. Implementation involves building the actual system with software by designing and programming software classes in accordance with the design specifications. The result should be a software implementation that meets the requirements that were identified during the requirements analysis phase. Development proceeds somewhat sequentially from requirements analysis to design specifications to implementation; however, the three phases may overlap. For example, it may help during requirements analysis to produce a prototype model for the user to test, which may help the user to discover and articulate system requirements. Phases may also provide feedback to previous phases. For example, a new requirement may be discovered during the implementation phase. A new requirement will subsequently affect requirements analysis and design specifications. Another software development consideration concerns evolution, which refers to the fact that the software will require future modifications. The user will discover software bugs, and the user's requirements will evolve over time. Both circumstances will require system modifications. The software developer will need to conduct the software development process or a portion of the process again. Therefore, the development process can be viewed as an iterative, cyclical process.

This thesis produced two software products. One product is a model for constructing agents, and the other product is an application that demonstrates how the model may be used. The following discussion on requirements analysis, design specification, and implementation concerns the development of both products. However, not all of the artifacts presented below pertain to both products. If the artifact does not pertain to both products, then this fact is indicated in the title of the table or the caption of

the figure that contains the artifact. If no such indication is given, then the artifact pertains to both products.

1. Requirements Analysis

The requirement analysis phase begins with an informal problem statement that describes the user's requirements. The initial problem statement is usually ambiguous, incomplete, contradictory, and may not reflect cost considerations; therefore, requirements analysis must turn the problem statement into a set of precise and measurable requirements [Ref. 9]. The problem statement for the ACSA Intelligent Agent is as follows:

> The system will automate some of the tasks involved in analyzing data contained in the ACSA database. The system will determine assessment scores by organizational safety area and as an overall assessment. The data may be horizontally fragmented by respondent and squadron attributes. The system will also look for patterns that identify anomalies in data entry or in question responses.

In order to develop precise requirements from the problem statement, several use cases were developed. A use case is a narrative that describes the sequence of events for an actor (an external agent) using a system to complete a process. Use cases do not define requirements, but they illustrate and imply requirements in the stories they tell. Functional requirements can then be developed from the use cases. The requirements analysis phase produced the following use cases:

Table 17: Start System Use Case

Use Case:	Start System
Actors:	User (initiator), System
Purpose:	To start the system
Overview:	The user specifies what set of rules to use and how often to analyze data in the database.
Actor Action	***System Response***
1. The user either double-clicks an icon or selects an icon from a menu.	2. The system validates the rule set, confirms its ability to access the database, and notifies the user that it is ready.

Table 18: Subset Database Use Case

Use Case:	Subset Database
Actors:	User (initiator), System
Purpose:	To specify what subset of data to use for assessment
Overview:	The user specifies what subset of data will be used for assessing aviation safety.
Actor Action	***System Response***
1. This user indicates to the system that he or she wants to specify the database subset.	2. The system presents a list of attributes from the database that represents the attributes of the respondents and the squadrons.
3. The user selects attributes from the list.	4. The system records the subset specification for later use.

Table 19: Specify System Actions Use Case

Use Case:	Specify System Actions
Actors:	User (initiator), System
Purpose:	To specify what actions the system will take
Overview:	The user specifies what actions the system will take.

Actor Action	*System Response*
1. The user indicates to the system that he or she wants to specify what actions the system is to take.	
	2. The system presents a set of actions that the system is capable of performing. • The set will include two types of actions. One type will be conducting safety assessments on the specified horizontal data subset. The other type will be conducting data integrity checking of the entire database. • The set of actions will also include the destination of results and the frequency of action. The results of action may be a report, which may be printed, saved, displayed, or emailed to someone.
3. The user specifies the actions for the system to take.	
	4. The system records the specified actions for later use.

Table 20: Run the System Use Case

Use Case:	Run the system
Actors:	User (initiator), System
Purpose:	To start the system
Overview:	The system is set to run in accordance with the actions that the user has previously specified.
Actor Action	***System Response***
1. This user indicates to the system that the system should act in accordance with the action that he or she specified earlier.	2. The system will execute actions in accordance with the user's specified action parameters.

The requirements analysis phase also produces a conceptual diagram of the system, which is displayed in Figure 9. The conceptual diagram is further refined throughout the development process. The design specification and implementation phases will gradually transform the conceptual diagram into a class diagram that is then implemented with actual software code.

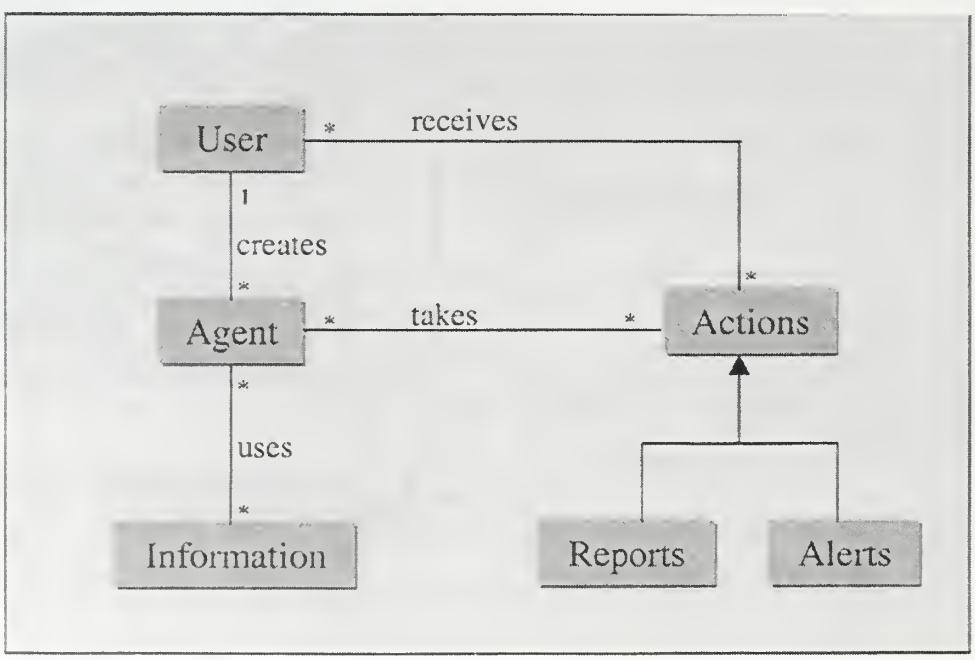

Figure 9: Conceptual Diagram

The artifacts produced during the requirements analysis phase help the developer to create a list of system requirements. The list of requirements will also include system constraints. Constraints may include time, money, legal, and hardware limitations. Each requirement is identified by a reference number. The reference number facilitates requirements management and tracking during future development phases and iterations. The requirements analysis phase for the ACSA Safety Agent produced the following list of requirements:

Table 21: System Requirements

Reference Number	Requirement
1	The system must use the same data that is contained in the Automated ACSA Survey System database. The system will not require any data that is not required for the ACSA survey. The system will use the same relational database management system that the Automated ACSA Survey System uses.
2	The system will access the database over a network. The system must be able to access the database, which may be behind a firewall.
3	The user must be able to subset the data horizontally by the respondent and squadron attributes.
4	The system will be able to dynamically adapt to a change in the survey questions. If the user modifies the survey by modifying the questions or the questionnaire, then the system will not need to be re-programmed to reflect the modification.
5	The user will be able to modify the rules and the models that the system uses to make inferences and conclusions about the data.
6	The system will be able to automatically send database analysis results to the user. Ideally, an event would trigger the action that notifies the user of the event. For example, if the system recognizes a significant anomaly or pattern during data analysis, then the user would be notified. The event may also be temporal, so that the action is triggered at a specified time.
7	The system must be modular in design, so individual components can be modified without affecting the rest of the system.
8	The system must comply with U.S. Navy and NPS network security policies.
9	The system must designed and documented in a manner that will assist future developers who will continue to develop the system. The code should be self-documenting, and useful header comments should be included for all classes and methods.
10	The system will detect database anomalies that may indicate corrupted

	data or otherwise erroneous data. See Requirement 5.
11	The system will produce summary assessments for aggregate records, which are included in the specified horizontal database subset. See Requirement 3.
12	The system must operate with the following minimum system requirements: • Windows 9X and NT • Pentium 200 MHz processor • 32 MB RAM • 28.8 Kbps modem

2. Design Specification

After determining the system requirements, the development process then enters the design specification phase, where a software design that satisfies the requirements is developed. The software should be specified in manner that is not specific to any particular programming language, but it is helpful to use the syntax of the target language if the target language is known in advance. The design specification for the ACSA Intelligent Agent uses Java syntax, because a goal of this research is to build the system using Java. However, the design specification could be used to implement the system in another programming language despite its Java appearance. Design specifications describe the system modules and the internal and external interfaces that will satisfy the system requirements. The design specification phase produces several artifacts that express the design of the system. The artifacts include a detailed architecture, class diagrams, state diagrams, and interaction diagrams. Figure 10 presents the application architecture.

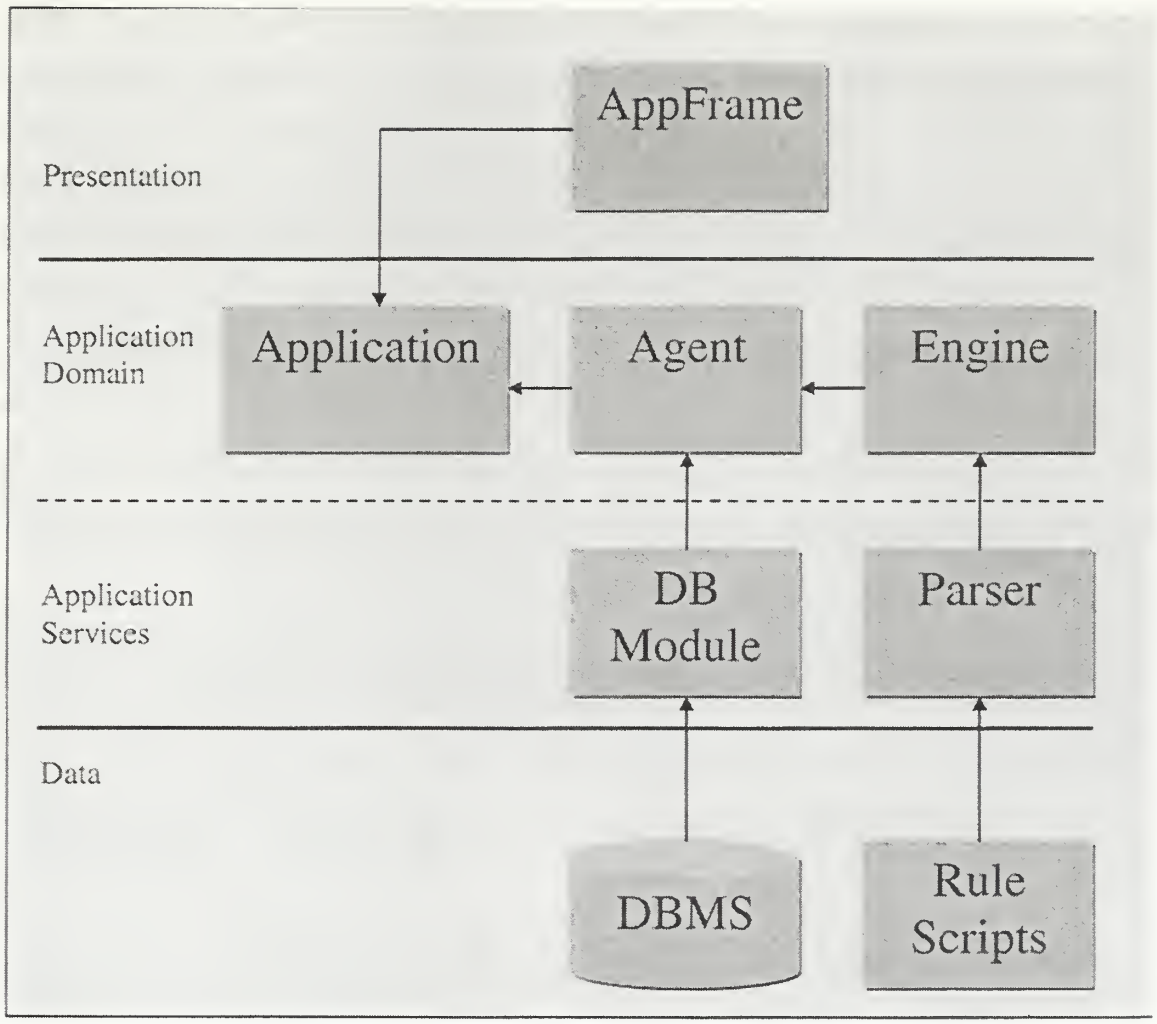

Figure 10: Application Architecture

3. Implementation

The system was implemented within the Java Programming Environment. The code was written primarily in the Java programming language, as specified in the Java 1.3 Release Candidate 1 Software Development Kit (JDK 1.3rc01). The inference engine was implemented with Jess 4.4, which is an expert system shell written in Java and is described in Chapter V. The system eventually was converted to run under Jess 5.0 beta. The model uses a commercial statistics package from Formanet for statistical

analysis. The statistics package is "com.formanet.math." Unfortunately, the use of this package is hard-wired into the model at this time. The agent uses statistical analysis as its only data mining modeling tool, so it was conveniently written into the agent model during implementation. Future development of the system should remove the statistical package from the agent model and include the statistical package in an application outside of the agent software package if the package is required. The following Java standard extensions were required for developing the application: JavaMail 1.1.3 and JavaBeans Activation Framework. Finally, the application uses a trial version of a commercial printing package, "com.wildcrest.j2textprinter," from Wildcrest, Inc. The classpath hierarchy is described in Figure 11. Each node beneath the "classpath" node in Figure 11 represents a Java package. The agentApp package contains the application, and the application package was included in the safetyAgent package solely for convenience.

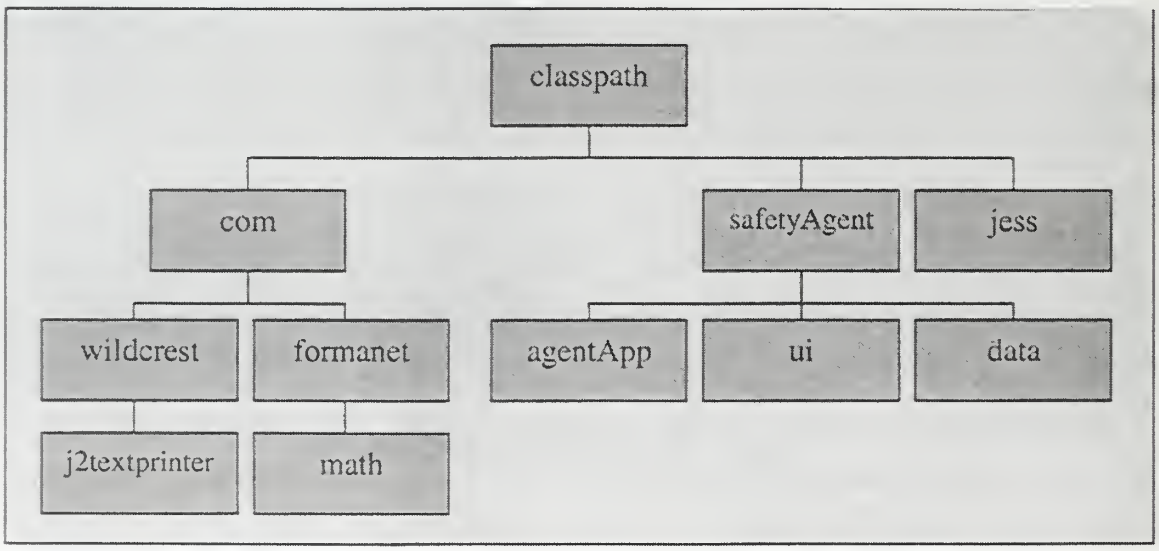

Figure 11: Classpath Hierarchy

C. AGENT MODEL

This thesis produced a model for building software assistants that can help researchers analyze the ACSA database. The model, which is shown in Figure 12, does

not do anything by itself. Instead, the model is a complete abstraction of the system, which describes a framework for building intelligent agents for ACSA database analysis.

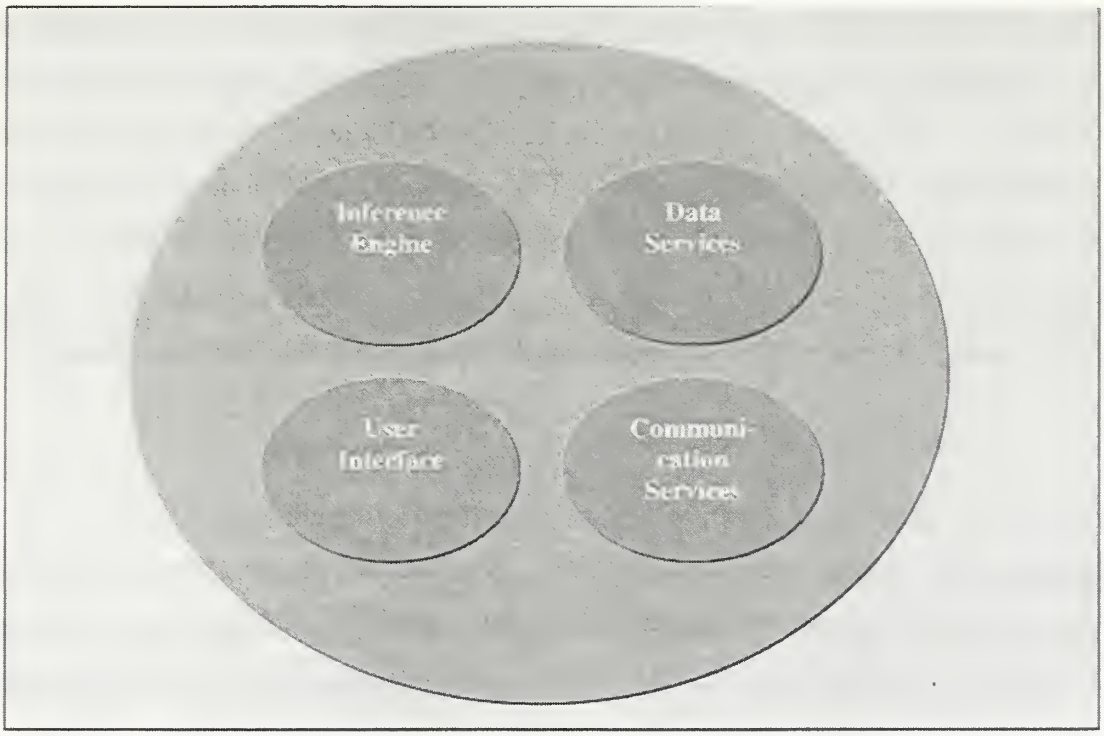

Figure 12: Agent Model

1. Inference Engine

The inference engine provides the agent with its reasoning capabilities. The inference engine may use conventional programming or artificial intelligence to infer knowledge from the data contained in the ACSA database. The type of programming style used depends upon the type of problems that the agent is trying to solve. Non-structured problems and semi-structured problems are best solved using artificial intelligence or expert system shell programming techniques. Conventional programming should be used for problems that are structured and have a known algorithm for solving. For example, determining the statistical mean and standard deviation for a set of numbers does not require artificial intelligence. However, making a recommendation for improving the safety climate of an organization may best be solved by using an expert

system shell. In practice, the inference engine will probably use a mix of conventional programming techniques and artificial intelligence.

An agent should use models for recognizing patterns and for making conclusions. For example, the agent could use a statistical model for recognizing statistical anomalies in the database, or the agent could use a predictive model for conducting predictive types of analysis. The agent could also use models for recommending mitigation or solutions to safety weaknesses that it recognizes in the data. Ideally, the inference engine should allow models to be inserted, removed, and modified. The inference engine could rely upon a model management system to assist in model management. Such an implementation may provide a cleaner design that is modular and easier to modify.

2. Data Services

The agent relies upon data and must have access to data. Therefore, the agent requires some kind of data service provider. The location, access means, and structure of the data will determine how the service is implemented. If all of the data is contained in a relational database, then a Java class that uses JDBC may be all that is required. However, if the data comes from multiple sources and combines structured and unstructured data, then a more sophisticated data access package may be required. The data may be internal or external to the system. The data may be contained in a database, in reports, in transaction processing systems, or in HTML pages on the World Wide Web or an intranet.

3. Communications Services

The nature and sophistication of communications services depends upon the required capabilities of the agent. If the agent is mobile, collaborates with other agents, and collects data from varied sources, then the agent may require a robust communications package. However, if the agent acts alone, is implemented in Java, and only requires access to a remote relational database, then no additional communication services outside of Java's built-in network application programming interfaces (API) is required.

4. User Interface

The agent will need to interact with the user, so the agent requires a user interface. The interface should use current human-computer interaction theory to create an interface that is relatively easy to use and meets the requirements of the system. The requirements may dictate a robust and powerful interface, or the requirements may dictate a simple "wizard" type of interface that walks the user through the interaction.

D. SAFETY AGENT PACKAGE

1. Safety Agent (safetyAgent.SafetyAgent)

a. Description

The safetyAgent.SafetyAgent class is the controlling class for the safety agent package. A developer builds an agent by creating an instance of the SafetyAgent class and then adding an Inference Engine instance, a DbModule instance, and an Options instance to the SafetyAgent instance via public set methods.

b. Public Interface

The SafetyAgent class provides only one other method outside of the public get and set methods required for its properties. This method is the loadFacts method, which takes no parameters and does not return a value. The loadFacts method uses the DbModule APIs to extract data from the database, apply statistical analysis models (particularly averages and standard deviations) to the data via the com.formanet.math package and conventional programming methods, and adds facts to the knowledge base.

2. Inference Engine (safetyAgent.InferenceEngine)

a. Description

The Inference Engine provides the reasoning capabilities for the agent. The Inference Engine must use some form of knowledge representation and reasoning

methodology. The safetyAgent.InferenceEngine implementation uses the Java Expert System Shell (Jess) from Sandia Laboratories. Jess is very similar to CLIPS, an expert system developed by NASA and written in the C language.

Jess uses production rules and facts as its knowledge base. Production rules are written as if-then rules. The "if" side of the if-then statement is called the conditional statement or the left-hand side (LHS). The "then" side of the if-then statement is called the conclusion or the right-hand side (RHS). The LHS is made up of facts, or conditional elements (CE), that form a "pattern." A pattern is matched if all conditional elements are true. Jess uses an algorithm known as the Rete Algorithm to match LHS conditional elements to facts in the knowledge base. The value of a CE may also be the return value of a function. For example, the CE "(test (< ?x 0))" calls a function that returns true if the value bound to variable ?x is less than zero. If a pattern matches the LHS of a rule, then the rule is "activated." See Figure 13 for a schematic view of the process.

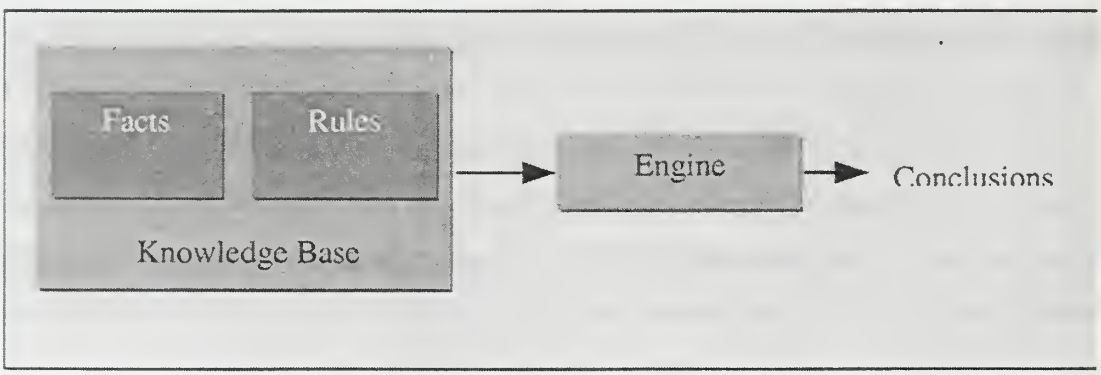

Figure 13: Making Conclusions

Although the rule has been activated, nothing will happen. The activated rule is simply placed on an agenda and waits for the engine to run. All activated rules will execute ("fire") their RHS when the inference engine runs. A rule can only fire once. To re-fire a rule, the facts of the LHS must be removed from the knowledge base and then re-asserted as facts into the knowledge base.

b. Public Interface

The inference engine is instantiated with a single parameter, which is a reference to the agent to which the engine belongs. The Inference Engine provides the following public methods:

- **void loadRules(InputStream rules)** – The method loads a Jess script from an input stream. The script must use Jess syntax, and it must declare all deftemplates, defclasses, and defrules for the Inference Engine.

- **jess.Value executeCommand(String command)** – Commands may be sent directly to Jess via this method. The command must be a valid Jess function or UserFunction. The return value is the same value that is returned by the function that is called by the command string.

- **void store(String name, Object object)** – The method provides a convenient way to pass Java objects to the Rete engine, so that Jess can access the object and its methods. The store method is the same as the Hashtable.put method. A stored object may be fetched in Jess and methods from the object's class may be called in Jess.

- **void addFact(Object fact, String templateName)** – The method adds a JavaBean to the knowledge base as a fact using the defclass that has the same name as the templateName string.

3. Data Module (safetyAgent.data.DbModule)

a. Description

The Data Module is defined as a Java Interface, so the implementing class will need to specify how data access is performed.

b. Public Interface

The following method signatures are declared by the Safety Module interface:

- **Map getAverageByQuestion(int version, DataSelectionCriteria[] criteria) throws SQLException** – Returns the average answer for each question in the questionnaire. The version integer is the version number for the questionnaire, and the criteria array describes the database horizontal subset.

- **DataSelectionCriteria[] getLongitudinalFields (int version) throws SQLException** – Enables the graphical user interface component to create the necessary controls for the user to specify the horizontal subset.

- **int[][] getOutcomes(int version, int questionNumber)** – Returns an array of all outcomes for the question specified by the questionNumber integer. Each row of the array contains a squadron number, a survey number, and the answer for that particular survey.

- **SafetyArea[] getSafetyAreas(int version)** – Returns an array of Safety Area instances, which, in turn, contain an array of all questions for that safety area. This method provides access to all of the survey questions in the database.

4. Questions and Safety Areas (safetyAgent.Question and safetyAgent.Safety Area)

a. Description

The Question and Safety Area classes are JavaBeans that describe the questions and safety areas in the database. The JavaBean properties for these classes include properties like question number, text, safety area name, average answer, and standard deviation. The Safety Agent class extracts data from the database via the Data Module class and creates Question and Safety Area JavaBeans. The Safety Agent class then adds the Question and Safety Area JavaBeans to the knowledge base. Jess creates "shadow facts" in the Rete engine that are bound to the JavaBeans. Jess uses reflection and the java.beans.PropertyChangeSupport class to dynamically change fact slot values to match the JavaBeans' properties. Each set method of a JavaBean must fire a property change event, in order to notify Jess that a JavaBean property has changed.

b. Public Interface

The public interfaces for the Question and Safety Area classes primarily consist of set and get methods. These classes provide a good example on how to use JavaBeans as facts in Jess. Both classes instantiate a java.beans.PropertyChangeSupport instance as an instance variable, and each get method calls the

PropertyChangeSupport.firePropertyChange method to notify all property change listeners that a property has been changed.

5. User Interface Package (safetyAgent.ui.*)

The User Interface Package contains implementations of the DataSelectorInterface and the QuestionSelectorInterface, which are defined in the safetyAgent.data package. These implementations have been included in order to help future developers to jumpstart safety agent applications. See Figure 17 for a screen shot of the DataSelectionPanel and the QuestionSelectionPanel. The QuestionSelectionPanel class implements the QuestionSelectorInterface, and the DataSelectionPanel class implements the DataSelectorInterface. These implementation classes primarily use JCheckBoxes for the user to select questions and to assign relative weightings per questions for assessment purposes and for enabling the user to specify the database subset. However, different classes can implement the interface by using different visual controls, such as, a JTree class instead of the JPanel and JCheckBox classes.

E. SAFETY AGENT APPLICATION

The safety agent application demonstrates how the Safety Agent package may be used to create an agent that can help researchers analyze the ACSA database. See Figure 2 above for a description of the application's architecture. The application instantiates and runs only one agent at a time. However, the Safety Agent package does not limit an application to only one agent. In fact, applications could theoretically instantiate and operate multiple agents simultaneously.

The application allows users to create, save, and re-use "profiles." A profile is a set of user preferences that specifies what the agent should do, where the agent should send the results, and when the agent should act.

The application can conduct two types of actions. The first action involves assessing the database for organizational safety. The second action involves inspecting the database for possible data entry errors. For each type of action, the user must specify when and how often the agent should conduct the action. The agent can act once, daily, weekly, or monthly. For each type of action, the user must also specify how the agent

should report the results of the action. The agent can send the results to the monitor, the printer, a file, a set of email addresses, or any combination of these. The agent presents the actions' results as summarized reports that are written in HTML.

The major system components of the application include the application code, the graphical user interface, the data module, the database, and the reports.

1. Safety Agent App (safetyAgent.agentApp.SafetyAgentApp)

The Safety Agent App class is the application and contains the main method for running the application. The Safety Agent App creates the agent, the database module, and the GUI. The application is run from the command line with the following command: java safetyAgent.agentApp.SafetyAgentApp.

2. Application Frame (safetyAgent.agentApp.AppFrame)

The Application Frame (Figure 14) controls the graphical user interface components and separates the GUI code from the application code. This class defines all actions that are associated with the menu and toolbar. Most actions simply call the corresponding method in the Safety Agent App class. However, if the Safety Agent App method requires input from the user, such as, a file name for saving the profile, then the Application Frame class will handle the user interaction required for getting the input from the user. Once the Application Frame has the user's input, the Application Frame instance will typically pass the input to the Safety Agent App instance in a method call.

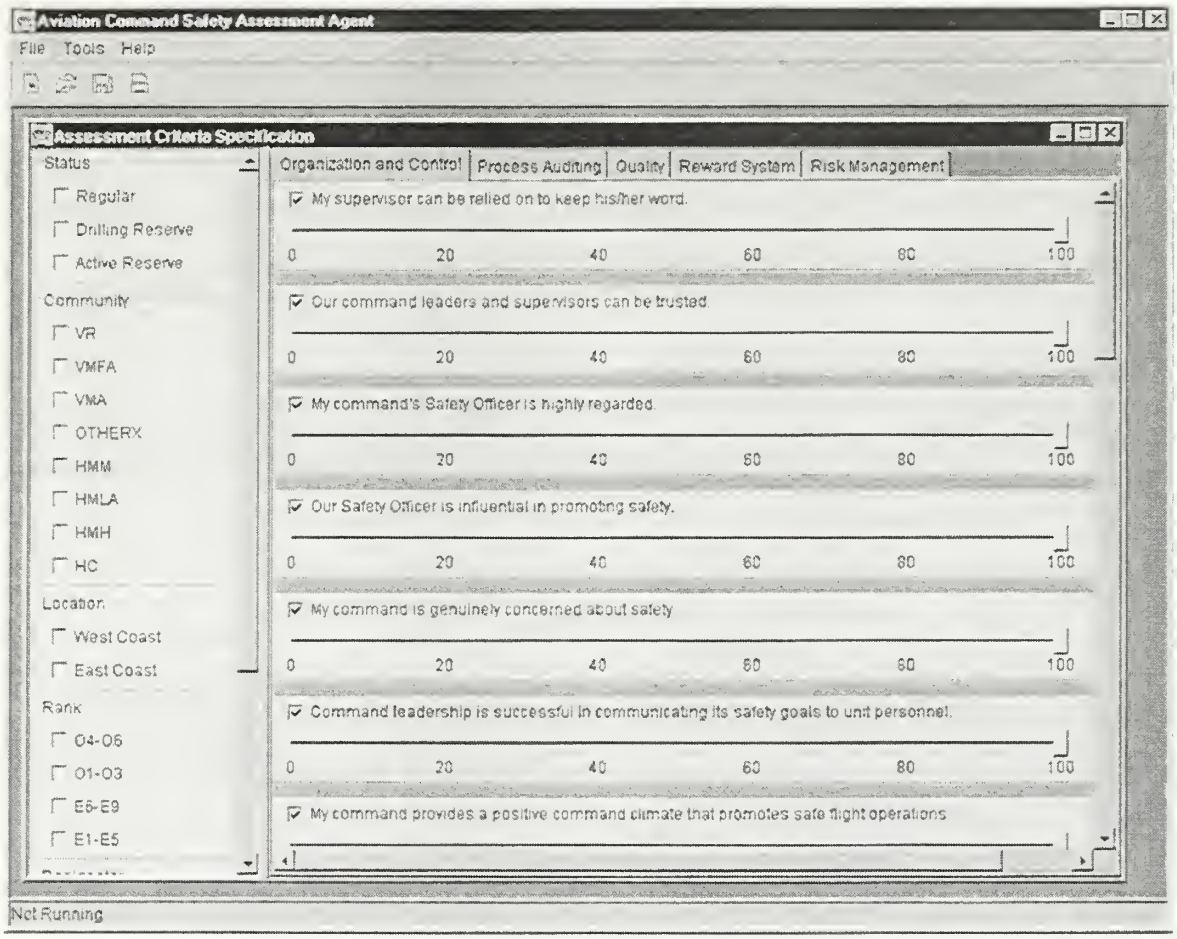

Figure 14: Application Frame

3. Data Module (safetyAgent.agentApp.DbModule_Survey3_DB)

The Data Module implements the safetyAgent.DbModule interface and provides data services for the agent by connecting and extracting data from the ACSA database via Java JDBC.

4. Database

The database structure and content is described in Chapter 3, "Database Design." The database can be either on the local machine or accessible via a network connection. The file, "safety_agent.properties," specifies the JDBC connection parameters. The

properties file is located in the safetyAgent\agentApp\resources directory. This file may be modified, but it should not be moved.

5. Rules

The rule file is located in the safetyAgent\agentApp\resources directory. The file defines 20 rules and is included in Appendix C. The file may be modified, but should not be moved. The following table displays the core rules from the rule file.

Table 22: Core Rules from Rule File

Number	Rule
1	(defrule weakness_in_safety_area "Weakness in a Safety Area" (doRule1) (safety_area (selected TRUE) (name ?safetyArea) (avgWtdAnswer ?average)) (test (< ?average 0)) => (assert(problem_exists_in_safety_area(safetyArea ?safetyArea))))
2	(defrule overall_weakness "Overall Weakness" (doRule2) (overall_wtd_average (wtdAverage ?average)) (test (< ?average 0)) => (assert(overall_problem_exists)))
3	(defrule inconsistency_in_safety_area "Inconsistent Outcome in a Safety Area" (doRule3) (question (safetyArea ?sa) (number ?q) (numDeviations ?sd)) (questionAssessmentThreshold (threshold ?t)) (test (>= ?sd ?t)) => (assert(inconsistent_outcome_within_safety_area (safetyArea ?sa) (questionNumber ?q))))
4	(defrule safety_area_not_consistent_with_overall "Safety area(s) is/are not consistent with the overall assessment" (doRule4) (safety_area (name ?sa) (numDeviations ?sd))

	(safetyAreaAssessmentThreshold (threshold ?t)) (test (>= ?sd ?t)) => (assert(safety_area_not_consistent_with_overall_average ?sa)))
5	(defrule negative_question_assessment "The question has a negative weighted average" (doRule6) (question (number ?n) (avgWtdAnswer ?a) (text ?txt)) (test (< ?a 0)) => (printout t "Question " ?n ", " ?txt ", has a negative assessment." crlf) (?*assessmentReport* addWeakQuestion ?n ?txt ?a))

6. Reports

The agent can produce two types of reports. First, the agent can produce a "Database Integrity Report." This report summarizes the results of the agent's inspection of the database for data entry anomalies. Second, the agent can produce a "Safety Assessment Report," which is shown in Figure 15.

The assessment report summarizes the results of the agent's organizational safety assessment analysis of either a subset of the database or the entire database. The agent can either display, email, print, or save the reports. Time events trigger the actions in this implementation of the safety agent application. The user specifies when and how often the agent should analyze the database and produce reports by setting the parameters in the Options Panel (Figure 16). Details concerning the reports and the models used to generate the reports follows in Subsection G below.

F. INSTALLATION AND OPERATION

1. Installation

The application runs on any platform that can run a Java Virtual Machine that supports Java Version 1.3rc01. The application also requires the following software:

- JavaMail 1.1.2 or higher
- JavaBeans Activation Framework (JAF)
- Wildcrest JTextPrinter 2.0
- Formanet Mathlite

JavaMail and JAF may be downloaded from the www.javasoft.com website. Both packages may be either installed as standard extensions in the jre\lib\ext directory or added to the system's classpath. Since JavaMail and JAF are standard extensions, Sun Microsystems recommends installing the packages as standard extensions. The application requires JTextPrinter for printing the reports. A trial version of JTextPrinter can be downloaded from www.wildcrest.com. The full retail version can be purchased for $99.95. Lastly, the application requires Mathlite for calculating averages and standard deviations. Mathlite can be purchased from www.formanet.com for $9.95. See Figure 11 above for an example classpath hierarchy for installing and running the application.

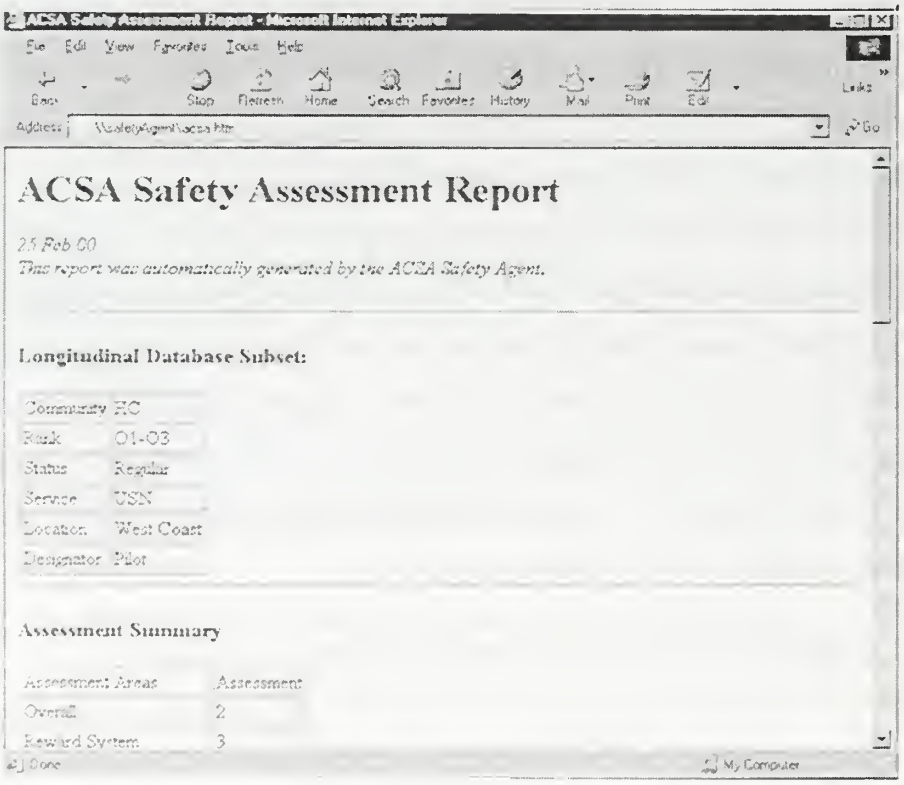

Figure 15: Assessment Report

2. Operation

a. Starting the Application

Start the application from the command line with the command: java safetyAgent.agentApp.SafetyAgentApp. This command assumes that the classpath is set properly. Instructions for setting the system classpath can be found at www.javasoft.com. The application may also be started from a batch file. Some IDEs, like Symantec Visual Café and IBM VisualAge, will compile Java source code into binary executables. However, neither method was tested, since native executable binary files are not part of the Java Programming Environment [Ref. 10].

b. Create a Profile

The application starts with a default profile. If the profile will not be saved, then simply select Tools → Options to modify the default options in the Options Panel (Figure 16). If the profile will be used again in the future, then it will be necessary to create a new profile by selecting File → New and following the dialog boxes.

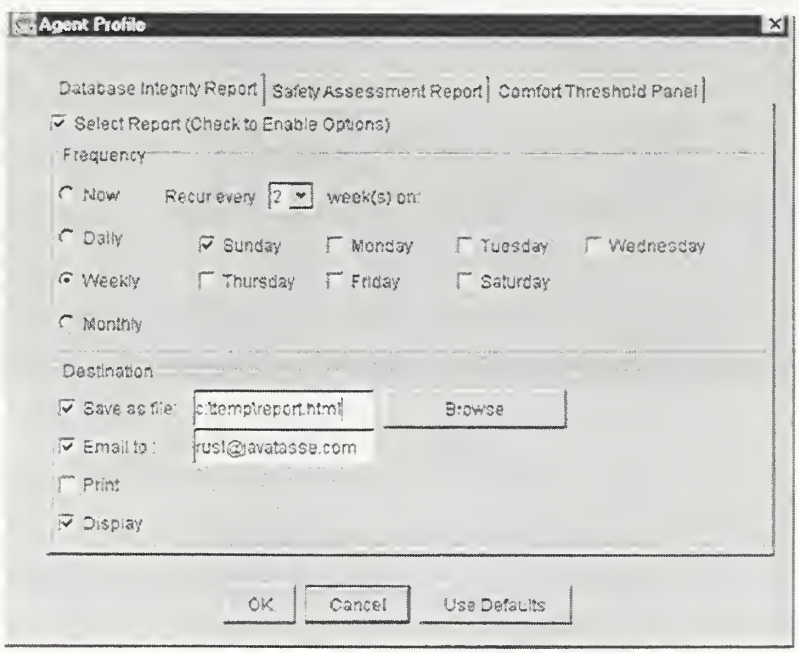

Figure 16 : Options Panel

c. Select Criteria and Questions

The application uses all records and all questions in the database by default. The database, however, may be fragmented horizontally and questions may be selected by selecting Tools → View Selector. The selector panel (Figure 17) displays the respondent and squadron attributes in the left-hand panel, and it displays the questions and safety areas in the right-hand panel. The relative importance or weighting of a question may be adjusted as well by adjusting the slider underneath each question.

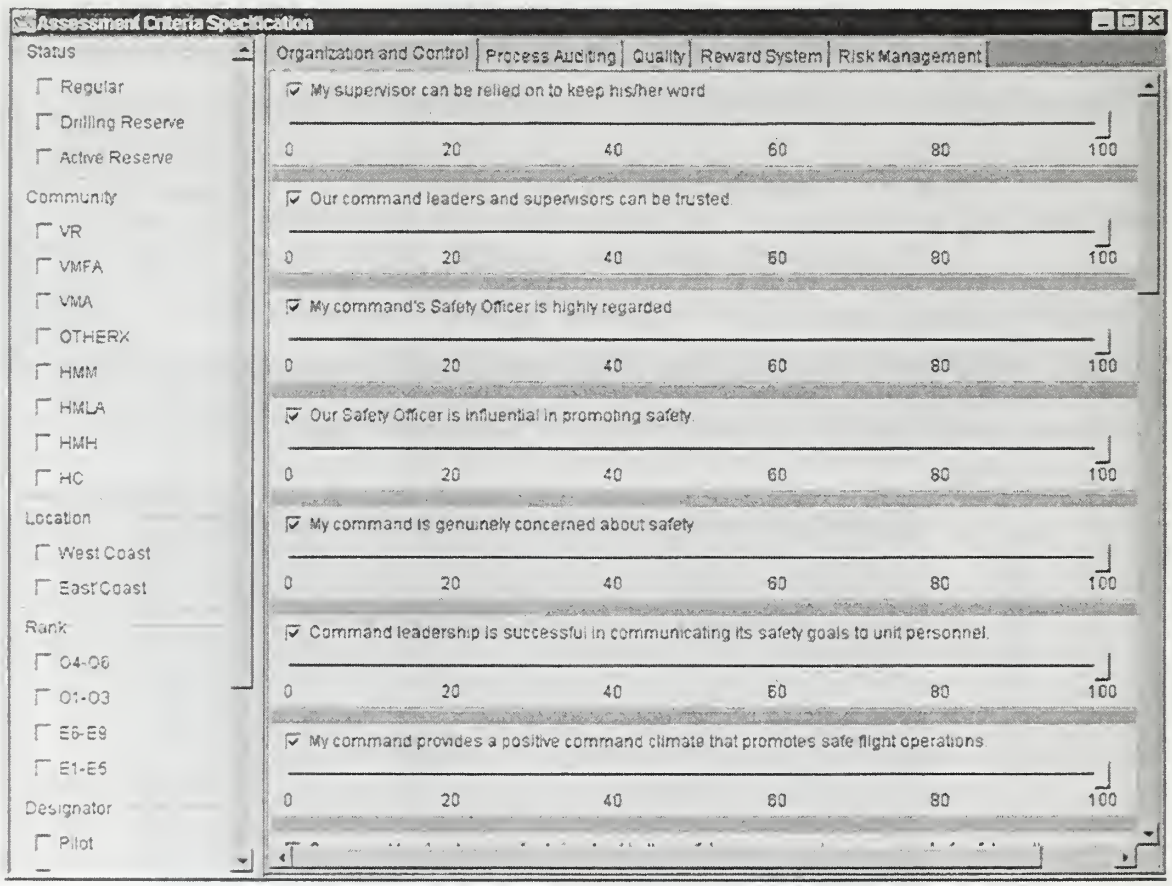

Figure 17: Selector Panel

G. ANALYSIS MODELS AND REPORTS

The initial development of the safety agent system did not develop sophisticated models for knowledge building. The research focused primarily on the creation of a software model for building agents. Future research may develop additional models for

the agent. For the demonstration application, however, a simple analysis model was developed and implemented.

The agent uses statistical analysis and production rules to create an analysis model for synthesizing knowledge from survey data. As mentioned previously, the statistical analysis model is unfortunately hard-coded in the SafetyAgent class. The SafetyAgent.loadFacts method creates JavaBeans for all questions and safety areas in the database. The method then extracts information from the database and sets the properties of the beans. Both the Question bean and the SafetyArea bean have properties for average answers and standard deviations. For safety assessment purposes, this is where the hard-coded model stops. The rule file contains the rest of the model for safety assessments. However, for database integrity analysis, the loadFacts method contains an algorithm for asserting that a survey answer is an anomaly. The method also contains an algorithm for asserting that a survey question or a respondent may deserve special attention by a researcher. This section will first discuss the model for safety assessment. Then this section will discuss the model for database integrity analysis. Lastly, this section will discuss the summary reports produced by the application.

1. Safety Assessment Model Implementation

The safety agent produces five safety assessments. First, the agent produces an overall score and determines if the overall assessment indicates a weak organizational safety climate. Second, the agent produces a score for each safety area and determines if any safety areas may need improvement. Third, the agent identifies safety areas that are inconsistent with the other safety areas. Fourth, the agent determines if any questions have received an overall negative response. Finally, the agent identifies questions that are inconsistent with the other questions related to the same safety area. The agent uses statistical means and standard deviations as model building blocks when conducting these five assessments. Model building blocks are functions and routines that may be used either alone or as building blocks for constructing larger models [Ref. 2]. In this case, the functions that produce the statistical analysis have been combined with conventional programming and expert system rules to construct a model for synthesizing safety assessment knowledge from raw data.

Two factors, however, complicate the use of raw ACSA survey data. First, some questions are negatively worded while others are positively worded. This fact means that the same answer for two questions may have the same meaning but may also result in an opposite evaluation for safety assessment purposes. A "strongly disagree" answer is considered a positive response for a negatively worded question, but "strongly disagree" is considered a negative response for a positively worded question. Therefore, although integer 0 means, "strongly disagree" for all questions, the ramification of integer 0 is not the same for all questions. Second, the ACSA Intelligent Agent application allows users to adjust the relative importance of each question. The straightforward way to implement this capability is to use a visual slider control (see Figure 17) that reduces the average answer for a question by a percentage. For example, if a question receives an average answer of 5 and its relative importance is 70 percent, then the average answer is reduced by 30 percent to 3.5. The problem here is that the reduction of the average answer inadvertently changes an "agree" answer into a "disagree" answer. Thus, the meaning of the respondent's answer has been changed, rather than having been reduced in significance.

The agent maps all responses to a common scale that maps the database values –1 through 6 to –3 through 3. This mapping eliminates the two problems discussed above. The following table describes the mappings:

Table 23: Survey Response Mappings

ACSA Likert Scale	Meaning	Positive Question Mapping	Negative Question Mapping
-1	N/A	0	0
0	Strongly Disagree	-3	3
1	Moderately Disagree	-2	2
2	Slightly Disagree	-1	1
3	Neutral	0	0
4	Slightly Agree	1	-1
5	Moderately Agree	2	-2
6	Strongly Agree	3	-3

The agent uses the statistical software package and conventional programming to determine the statistical means and standard deviations for questions, safety areas, and all questions together. Questions and safety areas are then asserted as facts into the knowledge base, and the Rete engine will fire all rules, in which the facts match patterns specified in the LHS. The rules in the knowledge base describe the model.

The rule file for the demonstration application contains the following rules concerning safety assessment (where X is an integer set by the user):

- If the overall assessment is less than zero, then assert that the overall assessment is weak.
- If a safety area assessment is less than zero, then assert that the safety area assessment is weak.
- If a question's average answer is less than zero, then assert that the question has received a negative assessment.
- If a safety area's assessment is X standard deviations outside of the statistical mean of all safety area assessments, then assert that the safety area is inconsistent.
- If a question's average answer is X standard deviations outside of the statistical mean of all other questions related to that question's safety area, then assert that the question is inconsistent with the other questions related to the same safety area.

2. Database Integrity Model Implementation

The safety agent produces three different conclusions concerning database integrity. First, the agent determines if any answer in the database is an anomaly. Second, the agent determines if any respondent should be considered more interesting than the others should. Third, the agent determines if any question should be considered more interesting than the others should. The agent uses the following model for determining these conclusions (X is defined by the user):

- If an answer to a question is X standard deviations from the mean for all responses for the same question, then this answer is an anomaly.
- If the database contains more than X anomalous answers for a question, then this question is interesting.

- If a respondent has entered more than X anomalous answers, then this respondent is interesting.

Unlike the if-then statements of the assessment analysis model, the if-then statements for the integrity analysis model are not written as rules in the rule file for the Inference Engine. Instead, the integrity analysis model is written as conventional if-then statements in the SafetyAgent.loadFacts method. However, the conclusions of the if-then statements are asserted as facts into the knowledge base, so that additional rules could be written to make further conclusions about the interesting questions and respondents.

3. Explanation of Summary Reports

The agent can produce an assessment report and a database integrity report. The AssessmentReport and IntegrityReport classes extend the abstract safetyAgent.Report class. Other concrete reports may be defined in future applications, but all must implement the following abstract methods: toString, toHtml, and toXml. All three methods return a String that represents the report in plain text, HTML, or XML respectively.

The Safety Agent App uses the toHtml method, so that the reports may be viewed in a JTextPane or in a web browser as formatted text. Figure 15 displays an assessment report as viewed in a web browser. Writing the reports in HTML makes formatting the report much easier than defining a custom StyledDocument class in Java. The report is sent in HTML and in plain text when the report is sent via email. The email is sent as a multipart MIME with content type set to "text/alternative." Each report is labeled "For Official Use Only" at the top and is entitled "Safety Assessment Report" or "Database Integrity Report" accordingly.

The assessment report includes the following sections:

- **Horizontal Subset** – the subset which was used to conduct the safety assessment.

- **Summary** – a table of assessment scores for the overall assessment and for each safety area. Scores ranges from –1 (very negative assessment) to +3 (very positive assessment).

- **Overall Negative Assessment** – if an overall negative assessment occurs, then this section indicates the negative overall assessment in red.

- **Negative Safety Areas** – a list of safety areas that received a negative assessment.

- **Inconsistent Safety Areas** – a list of safety areas that were inconsistent with the other safety area assessments.

- **Negative Questions** – a list of questions that received a negative assessment.

- **Inconsistent Questions** – a list of questions that were inconsistent with the other questions belonging to the same safety area.

The integrity report includes the following sections:

- **Interesting Questions** – a list of questions that the Inference Engine has determined to be the possible cause of database entry errors.

- **Interesting Respondents** – a list of survey respondents that the Inference Engine has determined to be the possible cause of database entry errors.

H. WEAKNESSES OF THE SAFETY AGENT PACKAGE

The Safety Agent package has two significant weaknesses, which should be improved in future development iterations. The model should be a complete abstraction of the system [Ref. 11]. The model depicted in Figure 12 is an abstract representation of the system, but the model, as implemented in the safetyAgent package, is not a complete abstraction.

The SafetyAgent.loadFacts method should not have hard-wired any analysis models into the system. Model management, insertion, deletion, and modification should be handled in an abstract manner, so that concrete implementations of safety agents can define their own models and model management sub-systems.

The InferenceEngine class is not an abstract class, and the class explicitly ties the safety agent's reasoning module to Jess, production rules, and the Rete algorithm. Therefore, the implementation precludes the use of other knowledge representation schemes and reasoning methods.

I. AREAS FOR FUTURE RESEARCH

1. Knowledge Engineering

The quality and sophistication of the agent's reasoning capabilities relies upon how well knowledge has been modeled and implemented. The first step in modeling knowledge is to elicit knowledge from from human experts. Knowledge engineering is defined as:

> The art of bringing the principles and tools of AI research to bear on difficult applications problems requiring experts' knowledge for their solutions. The technical issues of acquiring this knowledge, representing it and using it appropriately to construct and explain lines of reasoning are important problems in the design of knowledge-based systems. The art of constructing intelligent agents is both part of and an extension of the programming art. It is the art of building complex computer programs that represent and reason with knowledge of the world. [Ref. 12]

The analysis models and the rules for the agent must be improved. Future research could involve interviewing aviation safety experts and eliciting expert knowledge. Models and rules can then be built that express this expert knowledge and then added to a model package within the safetyAgent package. Likewise, future research could also involve developing a model management sub-system for the agent.

2. Agent Technology

The demonstration application currently deploys only one agent, but the safetyAgent package does not limit the number of agents that an application can deploy. Future research can explore how multiple agents could be used in the system. In fact, future research could explore many of the different facets of agent technology. Research could be conducted on mobile agents, collaborating agents, and learning agents. Future research could simply improve the rudimentary agent characteristics of the current agent.

3. Applications Development

The demonstration application could be used as the basis for developing a more robust ACSA Intelligent Agent application. The rule list could be extended to make use

of intermediate conclusions and to make conclusions that are more powerful. More actions, such as event-triggered alerts, could be implemented. The user interface could be studied and improved. The code, algorithms, and queries could be optimized to perform faster and more reliably.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Elmasri, Ramez and Shamkant, B. Navathe, Fundamentals of Database Systems, 2 ed., Addison-Wesley Publishing Co., 1994.

2. Turban, Efraim and Aronson, Jay E., Decision Support Systems and Intelligent Systems, 5th ed., Prentice Hall, 1998.

3. Byrd, T. A., "Expert Systems Implementation: Interviews with Knowledge Engineers," Industrial Management and Data Systems, v. 95, no. 10, 1995.

4. Giatarratano, Joseph and Riley, Gary, Expert Systems: Principles and Programming, 3rd ed., PWS Publishing Co., 1998.

5. Nwana, H. S., Software Agents: An Overview, Intelligent Systems Research AA&T, BT Laboratories; Ipswitch, 1996

6. Gilbert, Aparacio and others,"The Role of Intelligent Agents in Information Infrastructure," IBM, 1995.

7. Meyer, Marc H., "Risk Management in Financial Services: Current Applications of Technology and Business Reengineering."

8. [http://www.agentbuilder.com/AgentTools/index.html]

9. Berzins, Valdis, Software Engineering with Abstractions, Addison-Wesley Publishing Co., 1991.

10. [http://javasoft.com/index.html]

11. Alhir, Sinan Si, UML in a Nutshell, O'Reilly, 1998.

12. Feigenbaum, Edward and McCorduck, P., The Fifth Generation, Addison-Wesley Publishing Co., 1983.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. AGENT MODEL SOURCE CODE

A. SOURCE CODE: *safetyAgent.SafetyAgent.java*

```
/*
* Title:        SafetyAgent.java
* Version:      1.0
* Copyright:    Copyright (c)  All Rights Reserved.
* Author:       Rusl Flowers and David Dowler
* Project:      Thesis
* Description: This is the core class for creating and running
*               an ACSA Safety Agent.
*/

package safetyAgent;

import java.io.*;
import java.util.*;
import com.formanet.math.*;
import safetyAgent.data.*;


/**
 * This is the core class for creating and running an ACSA Safety
Agent.
 * The safety agent requires an Inference Engine, a DbModule, a
 * QuestionSelectionInterface, a DataSelectionInterface, an Options
instance
 * describing the user's preferences, and a Properties instance
 * containing an integer for "threshold.interesting_questions"
 * and an integer for "threshold.interesting_respondents".
 *
 * @author  Rusl Flowers
 * @author  David Dowler
 * @version 1.0
 */
public class SafetyAgent {

  /**
  * Default Constructor. Must set all of the agent's properties
  * before using. Otherwise, null pointer exceptions will result.
  */
  public SafetyAgent(){}

  /**
  * Constructs the agent with everything it needs to get the job done.
  */
  public SafetyAgent(DbModule dbm, QuestionSelectionInterface qsi,
    DataSelectionInterface dsi, Properties props, InferenceEngine
engine,
    Options defaultOptions, Options userOptions){
```

```
    setDbModule(dbm);
    setQuestionSelectionInterface(qsi);
    setDataSelectionInterface(dsi);
    setProperties(props);
    setInferenceEngine(engine);
    setDefaultOptions(defaultOptions);
    setUserOptions(userOptions);
  }

  public void setProperties(Properties props){
    properties = props;
  }

  public void setDbModule(DbModule dbModule){
    this.dbModule = dbModule;
  }

  public void setDataSelectionInterface(DataSelectionInterface dsi){
    this.dataSelector = dsi;
  }

  public void setQuestionSelectionInterface(QuestionSelectionInterface
qsi){
    this.questionSelector = qsi;
  }

  public void setInferenceEngine(InferenceEngine engine){
    this.engine = engine;
  }

  public void setUserOptions(Options options){
    userOptions = options;
  }

  public void setDefaultOptions(Options options){
    defaultOptions = options;
  }

  public Options getUserOptions(){
    return userOptions;
  }

  public DbModule getDbModule(){
    return dbModule;
  }

  public DataSelectionInterface getDataSelector(){
    return dataSelector;
  }

  public QuestionSelectionInterface getQuestionSelector(){
    return questionSelector;
  }
```

```
  public Options getDefaultOptions(){
    return defaultOptions;
  }

  public InferenceEngine getInferenceEngine(){
    return engine;
  }

  public void setRules(InputStream rules){
    engine.loadRules(rules);
  }

  public Report[] getReports(){
    return reports;
  }

  public void setReport(Report[] reports){
    this.reports = reports;
  }


  /**
  * Creates and loads facts into the inference engine's knowledge base.
  * Retrieves question and safety area information from the DbModule,
  * creates Question and SafetyArea beans, sets the beans' properties
  * based on the data from the DbModule and from statistical analysis,
and
  * then adds the beans as facts to the knowledge base.
  * <p>
  * Note: Questionnaire version number is set to 1. This was done to
save
  * time. Since there is currently only one questionnaire version and
no plans
  * to create a new version anytime soon, this built-in inflexibility
  * and ergo bad design was seen acceptable for rapid development.
  */
  public void loadFacts(){

    //debug
    System.out.println("\n" + new java.util.Date() + ": Loading
Facts...");

    try {

    engine.executeCommand(
      "(assert(questionAssessmentThreshold " +
        "(threshold " +
          userOptions.getQuestionAssessmentThreshold() + ")))");

    engine.executeCommand(
      "(assert(safetyAreaAssessmentThreshold " +
        "(threshold " +
          userOptions.getSafetyAreaAssessmentThreshold() + ")))");

    SafetyArea safetyArea[] = questionSelector.getSafetyAreas();
```

```
    /*
    *  The following two vectors are used for determining the
    *  statistical means and std. deviations for the overall
    *  weighted answer and scaled answer, respectively.
    */
    Vector overallWtdAnswerVector = new Vector();
    Vector overallScaledAnswerVector = new Vector();

    Map averageAnswers = dbModule.getAverageByQuestion(
      1, dataSelector.getDataSelectionCriteria());

    Hashtable questionAnomalies = new Hashtable();
    Hashtable respondentAnomalies = new Hashtable();



    for (int i = 0; i < safetyArea.length; i++){
    /*
    * All data is analyzed by safety area, so this is a double nested
loop
    * where the outer loop iterates through the safety areas and the
    * inner loop iterates through the questions for each safety area.
    * Furthermore, the inner loop is actually conducted twice. Once for
    * assessment and once for integrity checking.  The integrity
checking
    * loop contains the second nested loop, which iterates through all
of the
    * answers for a particular question.
    */

      SafetyArea sArea = safetyArea[i];
      Question[] questions = sArea.questions();

      //debug
      System.out.println("\nLoading Facts for " + sArea.getName() +
":");

      Vector avgScaledAnswerVector = new Vector();
      Vector avgWtdAnswerVector = new Vector();
      boolean selected = false;

      for (int j = 0; j < questions.length; j++){

        String qNumber = (new
Integer(questions[j].getNumber())).toString();
        float avg = ((Float)averageAnswers.get(qNumber)).floatValue();

        questions[j].setAvgAnswer( Math.round(avg));

        engine.addFact(questions[j], "question");

        avgScaledAnswerVector.add(
          new Double(questions[j].getAvgScaledAnswer()));
```

```
    if (questions[j].isSelected()){

      avgWtdAnswerVector.add(
        new Double(questions[j].getAvgWtdAnswer()));

      if (!sArea.isSelected()){
        sArea.setSelected(true);
      }
    }
  }  //end of first question loop

  sArea.setAvgScaledAnswer(Math.round((float)
    new FStatistics(avgScaledAnswerVector).getMean()));

  sArea.setStdDeviation(
    new FStatistics(avgScaledAnswerVector).getStandardDeviation());

  overallScaledAnswerVector.add(
    new Double(sArea.getAvgScaledAnswer()));

  if (avgWtdAnswerVector.size() > 0){
    sArea.setAvgWtdAnswer(
      new FStatistics(avgWtdAnswerVector).getMean());
    overallWtdAnswerVector.add(
      new Double(sArea.getAvgWtdAnswer()));
  }


  for (int k = 0; k < questions.length; k++){
  /*
  * Must loop through the questions again for integrity checking.
  * Integrity checking could have been done during the first loop,
  * but I separated it out because I will want to move this into a
  * another method later.
  */
    // debug
    System.out.print(
      "   integrity checking for question " +
      questions[k].getNumber() + " ");

    if(questions[k].isSelected() &&
       sArea.getStdDeviation() > 0){
      questions[k].setNumDeviations(
        (Math.abs(
          sArea.getAvgScaledAnswer() -
          questions[k].getAvgScaledAnswer()))/
          sArea.getStdDeviation());
    }

    int[][] outcomeTuples = dbModule.getOutcomes(
      1, questions[k].getNumber());

    Vector outcomes = new Vector();

    for (int p = 0; p < outcomeTuples.length; p++){
```

```
          outcomes.add(new Double(outcomeTuples[p][2]));
        }

        double expectedOutcome = new FStatistics(outcomes).getMean();
        double deviation = new
FStatistics(outcomes).getStandardDeviation();

        questions[k].setStdDeviation( deviation );

        // Loop through all of the outcomes for this question
        for (int q = 0; q < outcomeTuples.length; q++){

          double ans = outcomeTuples[q][2];
          double ansVariance = Math.abs(ans - expectedOutcome);
          double numDev = 0;

          if(deviation > 0){
            numDev = (ansVariance / deviation);
          }

          if (numDev >= userOptions.getIntegrityThreshold()){

            //debug
            System.out.print("*");  // creates a histograph for
anomalies

            int questionNumber = questions[k].getNumber();
            int surveyNumber   = outcomeTuples[q][0];
            int squadronNumber = outcomeTuples[q][1];

            engine.executeCommand(
              "(assert" +
                "(outcome_is_anomaly " +
                  "(questionNumber "  + questionNumber + ")" +
                  "(surveyNumber " + surveyNumber + ")" +
                  "(squadronNumber " + squadronNumber + ")))");

            String questionKey = Integer.toString(questionNumber);
            if (questionAnomalies.containsKey(questionKey)){
              Integer frequency1 =
(Integer)questionAnomalies.remove(questionKey);
              frequency1 = new Integer(frequency1.intValue() + 1);
              questionAnomalies.put(questionKey, frequency1);
            }
            else {
              questionAnomalies.put(questionKey, new Integer(1));
            }

            String respondentKey =
              Integer.toString(squadronNumber) + " " +
              Integer.toString(surveyNumber);
            if (respondentAnomalies.containsKey(respondentKey)){
              Integer frequency2 =
                (Integer)respondentAnomalies.remove(respondentKey);
              frequency2 = new Integer(frequency2.intValue() + 1);
```

```
            respondentAnomalies.put(respondentKey, frequency2);
          }
          else {
            respondentAnomalies.put(respondentKey, new Integer(1));
          }
        }
      } // end of outcomesTuple loop

      //debug
      System.out.println();

    } // end of second questions loop

    engine.addFact(sArea, "safety_area");

  }// end of safety areas loop

  Enumeration keys = questionAnomalies.keys();
  while (keys.hasMoreElements()){
    String key = (String)keys.nextElement();
    Integer freq = (Integer)questionAnomalies.get(key);
    if (freq.intValue() >=
        Integer.parseInt(properties.getProperty(
        "threshold.interesting_questions"))){
      engine.executeCommand(
        "(assert" +
          "(interesting_question_anomaly" +
            "(questionNumber " + key + ")" +
            "(frequency " + freq.intValue() + ")))");
    }
  }

  Enumeration keys2 = respondentAnomalies.keys();
  while (keys2.hasMoreElements()){
    String key2 = (String)keys2.nextElement();
    Integer freq2 = (Integer)respondentAnomalies.get(key2);
    if (freq2.intValue() >=
        Integer.parseInt(properties.getProperty(
        "threshold.interesting_respondents"))){
      StringTokenizer tk = new StringTokenizer(key2);
      int sqdrnNumber = Integer.parseInt(tk.nextToken());
      int svyNumber = Integer.parseInt(tk.nextToken());
      engine.executeCommand(
        "(assert" +
          "(interesting_respondent_anomaly" +
            "(surveyNumber " + svyNumber + ")" +
            "(squadronNumber " + sqdrnNumber + ")" +
            "(frequency " + freq2.intValue() + " )))");
    }
  }

  if (overallWtdAnswerVector.size() > 0){
    engine.executeCommand(
      "(assert(overall_wtd_average " +
        "(wtdAverage " +
```

```
            new FStatistics(overallWtdAnswerVector).getMean() + ")))");
    }

    double overallScaledAverage =
      new FStatistics(overallScaledAnswerVector).getMean();

    for (int m = 0; m < safetyArea.length; m++) {
      safetyArea[m].setNumDeviations(
        (Math.abs(
          safetyArea[m].getAvgScaledAnswer() -  overallScaledAverage))/
          new
FStatistics(overallScaledAnswerVector).getStandardDeviation());
    }

    } catch (java.sql.SQLException ex){
      ex.printStackTrace();
    }

    //debug
    System.out.println(new java.util.Date() + ":  Completed loading
facts.\n");
  }


  /**
  * Runs the Jess engine and resets the fact list afterwards.
  */
  public void runJess() {

    //debug
    System.out.println("\nThe engine is running.\n");

    engine.executeCommand("(assert(doRule1))");
    engine.executeCommand("(assert(doRule2))");
    engine.executeCommand("(assert(doRule3))");
    engine.executeCommand("(assert(doRule4))");
    engine.executeCommand("(assert(doRule5))");
    engine.executeCommand("(assert(doRule6))");

    ReportOptions assessRptOpt =
      userOptions.getAssessmentReportOptions();
    ReportOptions integrityRptOpt =
      userOptions.getDatabaseIntegrityReportOptions();

    ReportOptions[] rOption = {assessRptOpt, integrityRptOpt};
    String[] rptName = {"assessment", "integrity"};

    for (int i = 0; i < 2; i++){
    /*
    * Asserts the report options as facts into the
    * knowledge base.
    */

      ReportOptions opt = rOption[i];
      String rName = rptName[i];
```

```
      if (opt.getEmail()){
        String[] address = opt.getEmailAddress();
        StringBuffer command =
          new StringBuffer("(assert(email_" + rName + "_report(to ");
        for (int k = 0; k < address.length; k++){
          command.append( address[k] + " " );
        }
        command.append(")))");
        engine.executeCommand( command.toString() );
      }

      if (opt.getPrint()){
        engine.executeCommand("(assert(print_" + rName + "_report))");
      }

      if (opt.getSaveAsFile()){
        engine.executeCommand("(assert(save_" + rName + "_report))");
      }

      if (opt.getDisplayToScreen()){
        engine.executeCommand("(assert(display_" + rName +
"_report))");
      }
    }

    engine.executeCommand("(run)");

    System.out.println("\n\nResetting...clearing all facts");
    engine.executeCommand("(reset)");

  }

  private DbModule dbModule;
  private QuestionSelectionInterface questionSelector;
  private DataSelectionInterface dataSelector;
  private Properties properties;
  private InferenceEngine engine;
  private Options defaultOptions;
  private Options userOptions;
  private Report[] reports;
}
```

B. SOURCE CODE: *safetyAgent.InferenceEngine*

```
/*
* Title:        InferenceEngine.java
* Version:      1.0
* Copyright:    Copyright (c)  All Rights Reserved.
* Author:       Rusl Flowers and David Dowler
* Project:      Thesis
```

```
* Description: Conducts reasoning for the Safety Agent
*/

package safetyAgent;

import jess.*;
import java.io.*;
import javax.swing.*;


/**
 * Conducts reasoning for the Safety Agent.
 *
 * @author  Rusl Flowers
 * @author  David Dowler
 * @version 1.0
 */
public class InferenceEngine {

  private Rete rete;
  private SafetyAgent parent;

  public InferenceEngine( final SafetyAgent parent ) {

    this.parent = parent;

    rete = new Rete();

    try {
      rete.executeCommand(
        "(printout t \"The ACSA Agent is listening...\" crlf)");
      //rete.executeCommand("(watch all)");

      rete.addUserpackage(new MiscFunctions());
      rete.addUserpackage(new ReflectFunctions());
      //String scriptFile =
(Rete.class.getResource("scriptlib.clp")).getFile();
      //rete.executeCommand("(batch " + scriptFile + ")");
      rete.store("agent",parent);
    }
    catch(Exception e){
      e.printStackTrace();
    }
  }

  public void store( String name, Object obj ){
    rete.store(name, obj);
  }

  /**
  * Parses a Jess script into the engine
  *
  * @param rules   The Jess script
  */
  public void loadRules( InputStream rules ) {
```

```
    Jesp parser = new Jesp(new InputStreamReader(rules), rete);
    try{
      parser.parse(false);
    }
    catch (Exception e){
      e.printStackTrace();
    }
  }

  public Value executeCommand(String command){
    try{
      return(rete.executeCommand(command));
    }
    catch(JessException e){
      e.printStackTrace();
      Exception ex = (Exception)e.getNextException();
      if ( e != null){
        System.out.println("++++ Nested Exception +++++");
        ex.printStackTrace();
      }
      return null;
    }
  }

  /**
  * Adds an object, typically a JavaBean, to the fact list.
  *
  * @param fact           The bean
  * @param templateName   The deftemplate name for the bean
  */
  public void addFact(Object fact, String templateName){
    try {
      Funcall f = new Funcall("definstance", rete);
      f.add(new Value(templateName, RU.ATOM));
      f.add(new Value(fact));
      f.execute(rete.getGlobalContext());
    }
    catch (JessException e){
      System.err.println(e.getMessage());
      e.printStackTrace();
    }
  }

  /**
  * Pretty-prints the fact list
  */
  public String ppFacts(){
    return rete.ppFacts();
  }

}
```

C. SOURCE CODE: *safetyAgent.data.DbModule*

```
/*
* Title:        DbModule.java
* Version:      1.0
* Copyright:    Copyright (c)  All Rights Reserved.
* Author:       Rusl Flowers and David Dowler
* Project:      Thesis
* Description: Declares database services required for the Safety Agent
*/

package safetyAgent.data;

import java.util.*;
import java.sql.*;
import safetyAgent.*;

/**
* Declares database services required for the safety agent.
*
* @version 1.0
* @author Rusl Flowers
* @author David Dowler
*/
public interface DbModule {

  /**
  * Returns the safety areas and their questions
  */
  public SafetyArea[] getSafetyAreas(int version)
      throws SQLException;

  /**
  * The order of the returned array is guaranteed to be indexed
  * to the array of questions passed in.
  *
  * @param  questions  The questions
  * @param  criteria   Longitudinal subset of respondents. Can be null,
  *                    which will return the average for all
respondents.
  */
  public Map getAverageByQuestion( int version,
      DataSelectionCriteria[] criteria) throws SQLException;

  /**
  * Returns an array of arrays where, for each row, index 0
  * is the Survey_ID, index 1 is the Squadron_ID, and index 2
  * is the answer.
  *
  * @param  version         questionnaire version
  * @param  questionNumber  The question number
  */
```

```
  public int[][] getOutcomes(int version, int questionNumber)
    throws SQLException;

  /**
  * Returns any array of respondent and squadron attributes
  *
  * @param version  questionnaire version number
  */
  public DataSelectionCriteria[] getLongitudinalFields(int version)
    throws SQLException;
}
```

D. SOURCE CODE: *safetyAgent.Question*

```
/*
* Title:        Question.java
* Version:      1.0
* Copyright:    Copyright (c)  All Rights Reserved.
* Author:       Rusl Flowers and David Dowler
* Project:      Thesis
* Description:  Survey question bean. Encapslates all of the information
*               regarding a survey question. Jess uses the bean as a
fact.
*               Bean attributes are treated as slots by Jess, and
property
*               change support and listeners are added to enable Jess to
*               track any changes to the bean's attributes.
*/

package safetyAgent;

/**
* Survey question bean encapslates information
* regarding a survey question.  Jess uses the bean as a fact.
* Bean attributes are treated as slots by Jess, and property
* change support and listeners are added to enable Jess to
* track any changes to the bean's attributes.
*
* @version    1.0
* @author     Rusl Flowers
* @author     Dave Dowler
*/
public class Question {

  private int     number = 0;
  private String  text = "not assigned";
  private String  safetyArea = "not assigned";
  private float   weight;
  private boolean selected;
  private boolean positive = true;
  private int     avgAnswer = -1111; //-1111 means "never initialized"
  private double  stdDeviation;
```

```
private double  numDeviations;

public Question(){}

public Question(int num, String txt, boolean pos){
  setNumber(num);
  setText(txt);
  setPositive(pos);
}

public boolean isSelected(){
  return selected;
}
public float getWeight(){
  return weight;
}
public int getNumber(){
  return number;
}
public String getText(){
  return text;
}
public String getSafetyArea(){
  return safetyArea;
}
public boolean isPositive(){
  return positive;
}
public int getAvgAnswer(){
  return avgAnswer;
}
/**
* Maps Likert scale used by the ACSA Survey to a common scale
*/
public int getAvgScaledAnswer(){
  switch(avgAnswer){
    case 0:  return isPositive()? -3 :  3;
    case 1:  return isPositive()? -2 :  2;
    case 2:  return isPositive()? -1 :  1;
    case 4:  return isPositive()?  1 : -1;
    case 5:  return isPositive()?  2 : -2;
    case 6:  return isPositive()?  3 : -3;
    default: return  0;
  }
}
public double getAvgWtdAnswer(){
  return (getAvgScaledAnswer() * weight);
}
public double getStdDeviation(){
  return stdDeviation;
}
public double getNumDeviations(){
  return numDeviations;
}
```

```
  // Set Methods //
  // Fires property changes to notify Jess Rete engine of changes//

  public void setSelected(boolean newValue){
    boolean tmp = selected;
    selected = newValue;
    pcs.firePropertyChange("selected", tmp, selected);
  }

  public void setNumber(int newValue){
    int tmp = number;
    number = newValue;
    pcs.firePropertyChange("number", tmp, number);
  }

  public void setText(String newValue){
    String tmp = text;
    text = newValue;
    pcs.firePropertyChange("text", tmp, text);
  }

  public void setSafetyArea(String newValue){
    String tmp = safetyArea;
    safetyArea = newValue;
    pcs.firePropertyChange("safetyArea", tmp, safetyArea);
  }

  public void setWeight(float newValue){
    float tmp1 = weight;
    double tmp2 = getAvgWtdAnswer();
    weight = newValue;
    pcs.firePropertyChange("weight", new Float(tmp1),
                           new Float(weight));
    pcs.firePropertyChange("avgWtdAnswer", new Double(tmp2),
                           new Double(getAvgWtdAnswer()));
  }

  public void setPositive(boolean newValue){
    boolean tmp = positive;
    positive = newValue;
    pcs.firePropertyChange("positive", tmp, positive);
  }

  public void setAvgAnswer(int newValue){
    int tmp1 = getAvgScaledAnswer();
    int tmp2 = avgAnswer;
    double tmp3 = getAvgWtdAnswer();
    avgAnswer = newValue;
    pcs.firePropertyChange("avgScaledAnswer", tmp1,
getAvgScaledAnswer());
    pcs.firePropertyChange("avgAnswer", tmp2, avgAnswer);
    pcs.firePropertyChange("avgWtdAnswer", new Double(tmp3),
                           new Double(getAvgWtdAnswer()));
```

```
  }

  public void setStdDeviation(double newValue){
    double tmp = stdDeviation;
    stdDeviation = newValue;
    pcs.firePropertyChange("stdDeviation", new Double(tmp),
                           new Double(stdDeviation));
  }

  public void setNumDeviations(double newValue){
    double tmp = numDeviations;
    numDeviations = newValue;
    pcs.firePropertyChange("numDeviations", new Double(tmp),
                           new Double(numDeviations));
  }

  // Property Change Support and Listeners for Jess

  private java.beans.PropertyChangeSupport pcs =
    new java.beans.PropertyChangeSupport(this);

  public void
addPropertyChangeListener(java.beans.PropertyChangeListener pcl){
    pcs.addPropertyChangeListener(pcl);
  }
  public void
removePropertyChangeListener(java.beans.PropertyChangeListener pcl){
    pcs.removePropertyChangeListener(pcl);
  }
}
```

E. SOURCE CODE: *safetyAgent.SafetyArea*

```
/*
* Title:        SafetyArea.java
* Version:      1.0
* Copyright:    Copyright (c)  All Rights Reserved.
* Author:       Rusl Flowers and David Dowler
* Project:      Thesis
* Description: Organizational safety assessment area bean. Encapslates
*               safety assessment area information. Jess uses the bean
as a fact.
*               Bean attributes are treated as slots by Jess, and
property
*               change support and listeners are added to enable Jess to
*               track any changes to the bean's attributes.
*/

package safetyAgent;

import java.util.*;
```

```
/**
* Organizational safety assessment area bean. Encapsulates
* safety assessment area information. Jess uses the bean as a fact.
* Bean attributes are treated as slots by Jess, and property
* change support and listeners are added to enable Jess to
* track any changes to the bean's attributes.
*
* @version    1.0
* @author     Rusl Flowers
* @author     Dave Dowler
*/
public class SafetyArea {

  private String  name;
  private int     avgScaledAnswer;
  private double  avgWtdAnswer;
  private double  stdDeviation;
  private double  numDeviations;
  private boolean selected;
  private Vector  questions = new Vector();

  public SafetyArea(){
  }

  public SafetyArea(String name){
    setName(name);
  }

  public void addQuestion(Question q){
    questions.add(q);
  }

  public boolean isSelected(){return selected;}
  public String getName(){return name;}
  public int getAvgScaledAnswer(){return avgScaledAnswer;}
  public double getAvgWtdAnswer(){return avgWtdAnswer;}
  public double getNumDeviations(){return numDeviations;}
  public double getStdDeviation(){return stdDeviation;}
  public Question[] questions(){
    return (Question[])questions.toArray(new
Question[questions.size()]);}


  // Set Methods //
  // Fires property changes to notify Jess Rete engine of changes//

  public void setSelected(boolean newValue){
    boolean tmp = selected;
    selected = newValue;
    pcs.firePropertyChange("selected", new Boolean(tmp),
                           new Boolean(selected));
  }

  public void setName(String newValue){
```

```
    String tmp = name;
    name = newValue;
    pcs.firePropertyChange("name", tmp, name);
  }

  public void setAvgScaledAnswer(int newValue){
    int tmp = avgScaledAnswer;
    avgScaledAnswer = newValue;
    pcs.firePropertyChange("avgScaledAnswer", new Integer(tmp),
                           new Integer(avgScaledAnswer));
  }

  public void setAvgWtdAnswer(double newValue){
    double tmp = avgWtdAnswer;
    avgWtdAnswer = newValue;
    pcs.firePropertyChange("avgWtdAnswer", new Double(tmp),
                           new Double(avgWtdAnswer));
  }

  public void setNumDeviations(double newValue){
    double tmp = numDeviations;
    numDeviations = newValue;
    pcs.firePropertyChange("numDeviations", new Double(tmp),
                           new Double(numDeviations));
  }

  public void setStdDeviation(double newValue){
    double tmp = stdDeviation;
    stdDeviation = newValue;
    pcs.firePropertyChange("stdDeviation", new Double(tmp),
                           new Double(stdDeviation));
  }

  // Property Change Support and Listeners for Jess

  private java.beans.PropertyChangeSupport pcs =
    new java.beans.PropertyChangeSupport(this);

  public void
addPropertyChangeListener(java.beans.PropertyChangeListener pcl){
    pcs.addPropertyChangeListener(pcl);
  }
  public void
removePropertyChangeListener(java.beans.PropertyChangeListener pcl){
    pcs.removePropertyChangeListener(pcl);
  }
}
```

APPENDIX B. SAFETY AGENT APPLICATION SOURCE CODE

A. SOURCE CODE: *safetyAgent.agentApp.SafetyAgentApp*

```
/*
* Title:        SafetyAgentApp.java
* Version:      1.0
* Copyright:    Copyright (c)  All Rights Reserved.
* Author:       Rusl Flowers and David Dowler
* Project:      Thesis
* Description: The Safety Agent Application
*/

package safetyAgent.agentApp;

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import safetyAgent.ui.*;
import safetyAgent.data.*;
import safetyAgent.*;
import com.wildcrest.j2textprinter.*;
import javax.mail.*;
import javax.mail.internet.*;

/**
* The safety agent application.
*
* @version 1.0
* @author  Rusl Flowers
* @author  David Dowler
*/
public class SafetyAgentApp {

  private SafetyAgent agent = new SafetyAgent();
  private AppFrame frame;
  private JDialog optionsDialog;
  private Properties props = new Properties();
  private javax.swing.Timer timer;

  public SafetyAgentApp(){

    try {
      UIManager.setLookAndFeel(
        UIManager.getSystemLookAndFeelClassName());
    }
    catch (Exception e){
      e.printStackTrace();
    }
```

```
    //new Thread(new SplashFrame()).start();

    try {

      props.load(
        SafetyAgentApp.class.getResourceAsStream(
          "resources/safety_agent.properties"));

      agent.setProperties(props);
      agent.setDbModule(new DbModule_Survey3_DB(props));
      agent.setDataSelectionInterface(new DataSelectionPanel(agent));
      agent.setQuestionSelectionInterface(new
QuestionSelectionPanel(agent));
      agent.setInferenceEngine(new InferenceEngine(agent));
      agent.setRules(

SafetyAgentApp.class.getResourceAsStream("resources/rules.clp"));
      agent.setUserOptions(new Options());
      agent.setDefaultOptions(new Options());
      agent.getInferenceEngine().store("agentApp", this);

      frame = new AppFrame(this);
      frame.validate();
      frame.addWindowListener(new WindowAdapter() {
          public void windowClosing( WindowEvent e ){
            System.exit(0);
          }
      });

      //Center the window
      Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
      Dimension frameSize = frame.getSize();
      if (frameSize.height > screenSize.height){
        frameSize.height = screenSize.height;
      }
      if (frameSize.width > screenSize.width){
        frameSize.width = screenSize.width;
      }
      frame.setLocation((screenSize.width - frameSize.width) / 2,
        (screenSize.height - frameSize.height) / 2);

      frame.setVisible(true);

    }
    catch (IOException ioe){
      ioe.printStackTrace();
      JOptionPane.showMessageDialog
        (null, "Unable to load \"safety_agent.properties\" file.",
          "Error", JOptionPane.ERROR_MESSAGE);
      System.exit(0);
    }
    catch (ClassNotFoundException cnf){
      cnf.printStackTrace();
      JOptionPane.showMessageDialog
```

```
      (null, "Unable to find JDBC driver class. Check the " +
        "\"jdbc.driver\" property setting in the " +
        "\"safety_agent.properties\" file.",
        "Error", JOptionPane.ERROR_MESSAGE);
    System.exit(0);
  }

}

public static void main(String[] args) {
  SafetyAgentApp app = new SafetyAgentApp();
}

public SafetyAgent getAgent(){
  return agent;
}

public void runAgent(){


}

public void createProfile(){
}

public void openProfile( String fileName ) throws
    FileNotFoundException, IOException {

  Properties profile = new Properties();

  profile.load(new FileInputStream(fileName));

  String profileName = profile.getProperty("profileName");

  // Thresholds
  int integrityThreshold = Integer.parseInt(
    profile.getProperty("integrityThreshold"));
  int safetyAreaAssessmentThreshold = Integer.parseInt(
    profile.getProperty("safetyAreaAssessmentThreshold"));
  int questionAssessmentThreshold = Integer.parseInt(
    profile.getProperty("questionAssessmentThreshold"));

  // Assessment Report Options
  int aReport_frequency = Integer.parseInt(
    profile.getProperty("aReport_frequency"));
  int aReport_nthDay = Integer.parseInt(
    profile.getProperty("aReport_nthDay"));
  int aReport_weekDay = Integer.parseInt(
    profile.getProperty("aReport_weekDay"));
  int aReport_blankOfEveryMonth = Integer.parseInt(
    profile.getProperty("aReport_blankOfEveryMonth"));
  int aReport_ofEveryBlankMonths_1 = Integer.parseInt(
    profile.getProperty("aReport_ofEveryBlankMonths_1"));
  int aReport_ofEveryBlankMonths_2 = Integer.parseInt(
    profile.getProperty("aReport_ofEveryBlankMonths_2"));
```

```
    int aReport_blankDays = Integer.parseInt(
      profile.getProperty("aReport_blankDays"));
    int aReport_everyBlankWeeks = Integer.parseInt(
      profile.getProperty("aReport_everyBlankWeeks"));
    String[] aReportAddressArray = parseEmailAddresses(
      profile.getProperty("aReport_emailAddress"));
    String aReport_nameOfSaveFile =
      profile.getProperty("aReport_nameOfSaveFile");
    boolean aReport_email =
      new Boolean(profile.getProperty("aReport_email")).booleanValue();
    boolean aReport_print =
      new Boolean(profile.getProperty("aReport_print")).booleanValue();
    boolean aReport_saveAsFile =
      new
Boolean(profile.getProperty("aReport_saveAsFile")).booleanValue();
    boolean aReport_displayToScreen =
      new Boolean(
        profile.getProperty("aReport_displayToScreen")).booleanValue();
    boolean aReport_selected =
      new
Boolean(profile.getProperty("aReport_selected")).booleanValue();

    // Integrity Report Options
    int iReport_frequency = Integer.parseInt(
      profile.getProperty("iReport_frequency"));
    int iReport_nthDay = Integer.parseInt(
      profile.getProperty("iReport_nthDay"));
    int iReport_weekDay = Integer.parseInt(
      profile.getProperty("iReport_weekDay"));
    int iReport_blankOfEveryMonth = Integer.parseInt(
      profile.getProperty("iReport_blankOfEveryMonth"));
    int iReport_ofEveryBlankMonths_1 = Integer.parseInt(
      profile.getProperty("iReport_ofEveryBlankMonths_1"));
    int iReport_ofEveryBlankMonths_2 = Integer.parseInt(
      profile.getProperty( "iReport_ofEveryBlankMonths_2"));
    int iReport_blankDays = Integer.parseInt(
      profile.getProperty("iReport_blankDays"));
    int iReport_everyBlankWeeks = Integer.parseInt(
      profile.getProperty("iReport_everyBlankWeeks"));
    String[] iReportAddressArray = parseEmailAddresses(
      profile.getProperty("iReport_emailAddress"));
    String iReport_nameOfSaveFile =
      profile.getProperty("iReport_nameOfSaveFile");
    boolean iReport_email =
      new Boolean(profile.getProperty("iReport_email")).booleanValue();
    boolean iReport_print =
      new Boolean(profile.getProperty("iReport_print")).booleanValue();
    boolean iReport_saveAsFile =
      new
Boolean(profile.getProperty("iReport_saveAsFile")).booleanValue();
    boolean iReport_displayToScreen =
      new Boolean(
        profile.getProperty("iReport_displayToScreen")).booleanValue();
    boolean iReport_selected =
```

```
      new
Boolean(profile.getProperty("iReport_selected")).booleanValue();

    //-------------//
    // Set Options //
    //-------------//

    Options options = agent.getUserOptions();
    options.setIntegrityThreshold(integrityThreshold);

options.setSafetyAreaAssessmentThreshold(safetyAreaAssessmentThreshold)
;

options.setQuestionAssessmentThreshold(questionAssessmentThreshold);

    ReportOptions aOptions = options.getAssessmentReportOptions();
    aOptions.setSelected(aReport_selected);
    aOptions.setFrequency(aReport_frequency);
    aOptions.setBlankDays(aReport_blankDays);
    aOptions.setEveryBlankWeeks(aReport_everyBlankWeeks);
    aOptions.setNthDay(aReport_nthDay);
    aOptions.setWeekDay(aReport_weekDay);
    aOptions.setBlankOfEveryMonth(aReport_blankOfEveryMonth);
    aOptions.setOfEveryBlankMonths_1(aReport_ofEveryBlankMonths_1);
    aOptions.setOfEveryBlankMonths_2(aReport_ofEveryBlankMonths_2);
    aOptions.setSaveAsFile(aReport_saveAsFile);
    aOptions.setPrint(aReport_print);
    aOptions.setEmail(aReport_email);
    aOptions.setDisplayToScreen(aReport_displayToScreen);
    aOptions.setNameOfSaveFile(aReport_nameOfSaveFile);
    aOptions.setEmailAddress(aReportAddressArray);

    ReportOptions iOptions =
options.getDatabaseIntegrityReportOptions();
    iOptions.setSelected(iReport_selected);
    iOptions.setFrequency(iReport_frequency);
    iOptions.setBlankDays(iReport_blankDays);
    iOptions.setEveryBlankWeeks(iReport_everyBlankWeeks);
    iOptions.setNthDay(iReport_nthDay);
    iOptions.setWeekDay(iReport_weekDay);
    iOptions.setBlankOfEveryMonth(iReport_blankOfEveryMonth);
    iOptions.setOfEveryBlankMonths_1(iReport_ofEveryBlankMonths_1);
    iOptions.setOfEveryBlankMonths_2(iReport_ofEveryBlankMonths_2);
    iOptions.setSaveAsFile(iReport_saveAsFile);
    iOptions.setPrint(iReport_print);
    iOptions.setEmail(iReport_email);
    iOptions.setDisplayToScreen(iReport_displayToScreen);
    iOptions.setNameOfSaveFile(iReport_nameOfSaveFile);
    iOptions.setEmailAddress(iReportAddressArray);

    frame.setCurrentProfileName(profileName);

  }

  public void saveProfile( String fileName ) throws
```

```
      FileNotFoundException, IOException {

    Options options = agent.getUserOptions();

    Properties profile = new Properties();

    profile.setProperty("profileName", frame.getCurrentProfileName());
    profile.setProperty("integrityThreshold",
      Integer.toString(options.getIntegrityThreshold()));
    profile.setProperty("safetyAreaAssessmentThreshold",
      Integer.toString(options.getSafetyAreaAssessmentThreshold()));
    profile.setProperty("questionAssessmentThreshold",
      Integer.toString(options.getQuestionAssessmentThreshold()));

    ReportOptions aReport = options.getAssessmentReportOptions();
    ReportOptions iReport =
options.getDatabaseIntegrityReportOptions();

    String[] reportName = {"aReport", "iReport"};
    ReportOptions[] reportOption = {aReport, iReport};

    for (int i = 0; i <= 1; i++){

      profile.setProperty(reportName[i] + "_selected",
        new Boolean(reportOption[i].isSelected()).toString());
      profile.setProperty(reportName[i] + "_frequency",
        Integer.toString(reportOption[i].getFrequency()));
      profile.setProperty(reportName[i] + "_nthDay",
        Integer.toString(reportOption[i].getNthDay()));
      profile.setProperty(reportName[i] + "_weekDay",
        Integer.toString(reportOption[i].getWeekDay()));
      profile.setProperty(reportName[i] + "_blankOfEveryMonth",
        Integer.toString(reportOption[i].getBlankOfEveryMonth()));
      profile.setProperty(reportName[i] + "_ofEveryBlankMonths_1",
        Integer.toString(reportOption[i].getOfEveryBlankMonths_1()));
      profile.setProperty(reportName[i] + "_ofEveryBlankMonths_2",
        Integer.toString(reportOption[i].getOfEveryBlankMonths_2()));
      profile.setProperty(reportName[i] + "_blankDays",
        Integer.toString(reportOption[i].getBlankDays()));
      profile.setProperty(reportName[i] + "_everyBlankWeeks",
        Integer.toString(reportOption[i].getEveryBlankWeeks()));

      String[] emailAddresses = reportOption[i].getEmailAddress();
      StringBuffer buffer = new StringBuffer();

      for (int j = 0; j < emailAddresses.length; j++){
        buffer.append(emailAddresses[j]);
        if (j != (emailAddresses.length - 1)){
          buffer.append(";");
        }
      }

      profile.setProperty(reportName[i] +
"_emailAddress",buffer.toString());
      profile.setProperty(reportName[i] + "_nameOfSaveFile",
```

```
        reportOption[i].getNameOfSaveFile());
      profile.setProperty(reportName[i] + "_email",
        (new Boolean(reportOption[i].getEmail())).toString());
      profile.setProperty(reportName[i] + "_print",
        (new Boolean(reportOption[i].getPrint())).toString());
      profile.setProperty(reportName[i] + "_saveAsFile",
        (new Boolean(reportOption[i].getSaveAsFile())).toString());
      profile.setProperty(reportName[i] + "_displayToScreen",
        (new
Boolean(reportOption[i].getDisplayToScreen())).toString());
    }

    profile.store(
      new FileOutputStream(fileName),
      frame.getCurrentProfileName());

  }

  public void mail(Report report, String[] to){

    String from = props.getProperty("email.from");
    String subject = report.getTitle() + " " +
report.getDate().toString();
    Properties mailProps = new Properties();
    String smtpHost = props.getProperty("email.host");
    mailProps.put("mail.smtp.host", smtpHost);

    Session session = Session.getDefaultInstance(mailProps, null);
    session.setDebug(true);

    try {


      InternetAddress[] addresses = new InternetAddress[ to.length ];

      for ( int i = 0; i < to.length; i++){
        addresses[ i ] = new InternetAddress( to[i] );
      }

      Message message = new MimeMessage( session );
      message.setFrom( new InternetAddress( from ) );
      message.setRecipients(Message.RecipientType.TO, addresses);
      message.setSubject( subject );
      message.setSentDate( new Date() );

      MimeBodyPart mbp1 = new MimeBodyPart();
      mbp1.setContent(report.toString(), "text/plain");

      MimeBodyPart mbp2 = new MimeBodyPart();
      mbp2.setContent(report.toHtml(), "text/html");

      Multipart mp = new MimeMultipart("alternative");
      mp.addBodyPart(mbp1);
      mp.addBodyPart(mbp2);
```

```
      message.setContent(mp);

      Transport.send( message );
    }
    catch (MessagingException mex) {
      mex.printStackTrace();
    }

  }

  public void display(Report report){

    JTextPane pane = new JTextPane();

    pane.setContentType("text/html");
    pane.setEditable(false);
    pane.setText(report.toHtml());

    JFrame frame = new JFrame("report");
    JScrollPane sPane = new JScrollPane(pane);
    frame.getContentPane().add(sPane);
    frame.setSize(600, 700);
    frame.show();
  }

  public void print(Report report){

    JTextPane pane = new JTextPane();
    pane.setContentType("text/html");
    pane.setEditable(false);
    pane.setText(report.toHtml());

    J2TextPrinter j2tp = new J2TextPrinter();
    j2tp.setPrintDialogUsed(false);
    j2tp.setSeparatePrintThread(false);
    j2tp.setLeftMargin(1);
    j2tp.setRightMargin(1);
    j2tp.setTopMargin(1);
    j2tp.setBottomMargin(1);
    j2tp.setBoxAroundHeader(false);

    j2tp.print(pane);
  }

  public void save(Report report) {

    String fileName = "report.html";

    if ( report instanceof AssessmentReport ){
      fileName =
        agent.getUserOptions().getAssessmentReportOptions().
        getNameOfSaveFile();
    }
    else {
      fileName =
```

```
        agent.getUserOptions().getDatabaseIntegrityReportOptions().
        getNameOfSaveFile();
    }

    try {

      PrintWriter writer = new PrintWriter(new
FileOutputStream(fileName));
      writer.println(report.toHtml());
      if (writer.checkError()){
        System.out.println("Error writing file.");
      }
    }
    catch (IOException e){
      e.printStackTrace();
    }
  }

  public void exit(){
    System.exit(0);
  }

  private String[] parseEmailAddresses( String addressString ){

    StringTokenizer tk = new StringTokenizer(addressString, ";");
    int numberOfAddresses = tk.countTokens();
    String[] addressArray = new String[numberOfAddresses];

    try {
      for (int i = 0; i < numberOfAddresses; i++){
        addressArray[i] = tk.nextToken();
      }
    }
    catch(NoSuchElementException nse){
      nse.printStackTrace();
    }

    return addressArray;
  }

}
```

B. SOURCE CODE: *safetyAgent.agentApp.DbModule_Survey3_DB*

```
/*
* Title:       DbModule_Survey3_DB.java
* Version:     1.0
* Copyright:   Copyright (c)  All Rights Reserved.
* Author:      Rusl Flowers and David Dowler
* Project:     Thesis
* Description: Provides database services for the Safety Agent
*/
```

```
package safetyAgent.agentApp;

import java.sql.*;
import java.util.*;
import safetyAgent.*;
import safetyAgent.data.*;


/**
* Provides database services for the Safety Agent
*
* @author  Rusl Flowers
* @author  David Dowler
* @version 1.0
*/
public class DbModule_Survey3_DB implements DbModule {

  private static String url;
  private static String user;
  private static String password;

  public DbModule_Survey3_DB(Properties p) throws
ClassNotFoundException {

    Class.forName(p.getProperty("jdbc.driver"));
    url = p.getProperty("jdbc.url");
    user = p.getProperty("jdbc.user");
    password = p.getProperty("jdbc.password");
  }

  /**
  * Returns the safety areas and questions from the database
  * Defaults to version number 1, so the parameter is not used in
  * this version.
  */
  public final SafetyArea[] getSafetyAreas(int version)
    throws SQLException{

    String fromClause =
        "FROM Questions INNER JOIN Questions_Questionnaires ON " +
        "Questions.Question_ID = Questions_Questionnaires.Question_ID
";

    String whereClause =
"Questions_Questionnaires.Questionnaire_Version = 1 ";

    String safetyAreaQuery =
        "SELECT Questions.Safety_Area " + fromClause +
        " GROUP BY Questions.Safety_Area, " +
        " Questions_Questionnaires.Questionnaire_Version " +
        " HAVING " + whereClause;

    Vector tempV = new Vector();
    Connection conn = getConnection();
```

```
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(safetyAreaQuery);

    while(rs.next()){

      String safetyAreaName = rs.getString(1);
      SafetyArea sa = new SafetyArea(safetyAreaName);

      Statement stmt2 = conn.createStatement();

      ResultSet rs2 = stmt2.executeQuery(
        "SELECT Questions.Question_ID, Questions.Question_Text, " +
        " Questions.Is_Positive " + fromClause +
        " WHERE " + whereClause + " AND " +
        " Questions.Safety_Area = '" + safetyAreaName + "'");

      while(rs2.next()){

        Question q = new Question(
            rs2.getInt(1), rs2.getString(2), rs2.getBoolean(3));
        q.setSafetyArea( safetyAreaName );
        sa.addQuestion(q);
      }

      tempV.add(sa);
    }

    conn.close();

    return (SafetyArea[])tempV.toArray(new SafetyArea[tempV.size()]);
  }

  public final Map getAverageByQuestion(int version,
    DataSelectionCriteria[] criteria) throws SQLException{


    StringBuffer query =  new StringBuffer(
      "SELECT Answers.Question_ID, Avg(Answers.Answer) AS AvgOfAnswer "
+
      "FROM Respondents INNER JOIN " +
        "(Squadrons INNER JOIN " +
          "(Answers INNER JOIN " +
            "Questions_Questionnaires ON Answers.Question_ID = " +
            "Questions_Questionnaires.Question_ID) " +
          "ON Squadrons.Squadron_ID = Answers.Squadron_ID) " +
        "ON (Respondents.Squadron_ID = Answers.Squadron_ID) " +
            "AND (Respondents.Survey_ID = Answers.Survey_ID) ");


    query.append(" GROUP BY Answers.Question_ID, " +
        "Questions_Questionnaires.Questionnaire_Version ");

    if (criteria != null){

      Vector tempV = new Vector();
```

```
      for (int i = 0; i < criteria.length; i++){

        String fieldName =
            criteria[i].getTableName() + "." +
criteria[i].getFieldName();

        if (tempV.contains(fieldName)){
          continue;
        }
        else {
          tempV.add(fieldName);
          query.append(", ");
          query.append(fieldName);
          query.append(" ");
        }
      }
    }

    StringBuffer havingClause = new StringBuffer(
        " HAVING (((Questions_Questionnaires.Questionnaire_Version)=1)
");

    if (criteria != null){

      for (int i = 0; i < criteria.length; i++){
        havingClause.append(" AND ");
        havingClause.append( " ((" +
        criteria[i].getTableName() + "." + criteria[i].getFieldName() +
            ") = '" + criteria[i].getEqualTo() + "')");
        havingClause.append(" ");
      }
    }

    query.append(havingClause + ")");

    Map result = new TreeMap();

    Connection conn = getConnection();

    Statement stmt = conn.createStatement();

    ResultSet rs = stmt.executeQuery(query.toString());

    while (rs.next()){
      result.put(
        rs.getString("Question_ID"), new
Float(rs.getFloat("AvgOfAnswer")));
    }

    return result;
  }


  public final int[][] getOutcomes(
```

```
    int version, int questionNumber) throws SQLException{

    Connection conn = getConnection();
    Statement pstmt = conn.createStatement();
    Vector tempV = new Vector();

    ResultSet rs = pstmt.executeQuery(
      "SELECT Answers.Survey_ID, Answers.Squadron_ID, Answers.Answer "
+
      "FROM (Questions INNER JOIN Answers ON " +
      "Questions.Question_ID = Answers.Question_ID) INNER JOIN " +
      "Questions_Questionnaires ON Questions.Question_ID = " +
      "Questions_Questionnaires.Question_ID " +
      "WHERE (((Answers.Question_ID)= " + questionNumber + ") AND " +
      "((Questions_Questionnaires.Questionnaire_Version)= " + version +
"))");

    while(rs.next()){
      tempV.add(new int[]{rs.getInt(1), rs.getInt(2), rs.getInt(3)});
    }

    return (int[][])tempV.toArray(new int[tempV.size()][3]);
  }


  /**
  * Implements DbModule interface, and only returns fields, to which
  * a Respondent record is related.  For example, if no
  * respondent is a Marine, then USMC will not be included in the
  * fields.
  */
  public final DataSelectionCriteria[] getLongitudinalFields(int
version)
    throws SQLException {

    Vector result = new Vector();

    String[] squadronFields = {"Community", "Location"};
    String[] respondentFields = {"Rank", "Designator", "Status",
"Service"};

    Connection conn = getConnection();

    for (int i = 0; i < squadronFields.length; i++){

      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery(
        "select " + squadronFields[i] + " from Squadrons inner join " +
        "Respondents on Squadrons.Squadron_ID = Respondents.Squadron_ID
" +
        "group by " + squadronFields[i]);

     // v.add(squadronFields[i]);

      while(rs.next()){
```

```
        result.add(new DataSelectionCriteria(
          squadronFields[i], "Squadrons", rs.getString(1), false));
      }
      rs.close();
    }

    for (int i = 0; i < respondentFields.length; i++){

      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery(
        "select " + respondentFields[i] + " from Respondents inner join
" +
        "Squadrons on Squadrons.Squadron_ID = Respondents.Squadron_ID "
+
        "group by " + respondentFields[i]);

     // v.add(respondentFields[i]);

      while(rs.next()){
        result.add(new DataSelectionCriteria(
          respondentFields[i], "Respondents", rs.getString(1), false));
      }

      rs.close();
    }

    conn.close();

    return (DataSelectionCriteria[])result.toArray(
        new DataSelectionCriteria[result.size()]);
  }

  private static final Connection getConnection() throws SQLException {
    return DriverManager.getConnection(url,user,password);
  }
}
```

APPENDIX C. JESS RULE FILE

```
; Title: rules.clp
; Version:
; Author: Rusl Flowers and David Dowler
; Description: Defines the rules for the Safety Agent App


;;
; load JavaBean class definitions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defclass question safetyAgent.Question)
(defclass safety_area safetyAgent.SafetyArea)

;;
; Defglobals
;;;;;;;;;;;;

(defglobal ?*assessmentReport* = (new safetyAgent.AssessmentReport))
(defglobal ?*integrityReport* = (new safetyAgent.IntegrityReport))

;;
; Deftemplates
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftemplate safetyAreaAssessmentThreshold
  (slot threshold))

(deftemplate questionAssessmentThreshold
  (slot threshold))

(deftemplate overall_wtd_average
  "The overall average for weighted averages"
  (slot wtdAverage))

(deftemplate problem_exists_in_safety_area
  "A problem exists in a safety area"
  (slot safetyArea))

(deftemplate inconsistent_outcome_within_safety_area
    (slot safetyArea)
    (slot questionNumber))

(deftemplate outcome_is_anomaly
    (slot questionNumber)
    (slot surveyNumber)
    (slot squadronNumber))

(deftemplate interesting_question_anomaly
    (slot questionNumber)
```

```
    (slot frequency))

(deftemplate interesting_respondent_anomaly
    (slot surveyNumber)
    (slot squadronNumber)
    (slot frequency))

(deftemplate questionAssessmentThreshold
  (slot threshold))

(deftemplate safetyAreaAssessmentThreshold
  (slot threshold))

(deftemplate email_assessment_report
  (multislot to))

(deftemplate email_integrity_report
  (multislot to))


;;
;  Rule 1: Weakness in a Particular Safety Area
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;
; If doRule is true and a selected safety area's average weighted
answer
; is less than zero, then assert that a problem exists in that
particular
; safety area.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;

(defrule weakness_in_safety_area "Weakness in a Safety Area"
  (doRule1)
  (safety_area
    (selected TRUE)
    (name ?safetyArea)
    (avgWtdAnswer ?average))
  (test (< ?average 0))
  =>
  (assert(problem_exists_in_safety_area(safetyArea ?safetyArea))))

;;
; If a problem exists in a safety area, then print to the screen "The
; average weighted answer for safety area X is Y. Therefore, a safety
; problem may exist in safety area X.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule safety_area_problem_action "Do this if a problem is found"
  (problem_exists_in_safety_area
    (safetyArea ?sa))
  (safety_area
    (name ?sa)
    (avgWtdAnswer ?avg))
```

```
  =>
  (printout t "The average weighted answer for " ?sa " is " ?avg crlf)
  (printout t "Therefore, a safety problem may exist in " ?sa)
  (?*assessementReport addWeakArea ?sa ?avg))


;;
;  Rule 2: Overall Weakness
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;
; If doRule is true and the overall assessment is less than zero, then
; assert that an overall problem exists.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule overall_weakness "Overall Weakness"
  (doRule2)
  (overall_wtd_average
    (wtdAverage ?average))
  (test (< ?average 0))
  =>
  (assert(overall_problem_exists)))

;;
; If an overall problem exists, then print to the screen, "The overall
; average is X. Therefore, a general safety problem may exist."
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;

(defrule overall_problem_action "Do this if an overall problem is
found"
  (overall_problem_exists)
  (overall_wtd_average
    (wtdAverage ?average))
  =>
  (printout t "The overall average is " ?average crlf)
  (printout t "Therefore, a general safety problem may exist" crlf)
  (?*assessmentReport* addWeakOverall ?average))


;;
;  Rule 3: Inconsistent Outcome within a Safety Area
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;
; If doRule3 is true and a question's average scaled answer is two or
; more standard deviations from its safety area's average scaled
answer, then
; assert that the question is inconsistent with the other questions
; in that particular safety area.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
```

```
(defrule inconsistency_in_safety_area "Inconsistent Outcome in a Safety
Area"
  (doRule3)
  (question
     (safetyArea ?sa)
     (number ?q)
     (numDeviations ?sd))
  (questionAssessmentThreshold
    (threshold ?t))
  (test (>= ?sd ?t))
  =>
  (assert(inconsistent_outcome_within_safety_area
    (safetyArea ?sa)
    (questionNumber ?q))))

;;
; If a question is inconsistent with the other questions in its safety
area,
; then print to the screen, "Question X is inconsistent with the other
; questions in safety area Y."
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;

(defrule inconsistent_outcome_in_safety_area_action "Do this if a
question
   is inconsistent with its safety area's average answer"
  (inconsistent_outcome_within_safety_area
    (safetyArea ?sa)
    (questionNumber ?q))
  (question
    (number ?q)
    (text ?txt)
    (avgWtdAnswer ?q_avg))
  (safety_area
    (name ?sa)
    (avgWtdAnswer ?sa_avg))
  =>
  (printout t
    "Question " ?q "'s average is inconsistent with other questions in
" ?sa crlf)
  (?*assessmentReport* addInconsistentQuestion ?q ?txt ?q_avg ?sa
?sa_avg))


;;
;  Rule 4: Safety Area(s) not Consistent with Overall Average
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;
; If doRule4 is true and a safety area's average scaled answer is two
or
; more standard deviations from the overall average scaled average,
then
```

```
; assert that the safety area's average outcome is inconsistent with
the
; overall average outcome
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;

(defrule safety_area_not_consistent_with_overall
  "Safety area(s) is/are not consistent with the overall assessment"
  (doRule4)
  (safety_area
    (name ?sa)
    (numDeviations ?sd))
  (safetyAreaAssessmentThreshold
    (threshold ?t))
  (test (>= ?sd ?t))
  =>
  (assert(safety_area_not_consistent_with_overall_average ?sa)))

;;
; If a safety area's average outcome is inconsistent with the overall
; average outcome, then print to the screen, "Safety Area X is not
consistent
; with the overall assessment."
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;

(defrule inconsistent_safety_area_action
  "Do this if a safety area is inconsistent with the overall average"
  (safety_area_not_consistent_with_overall_average ?sa)
  (safety_area
    (name ?sa)
    (avgWtdAnswer ?avg))
  =>
  (printout t "Safety Area "  ?sa " is not consistent with the
    overall assessment." crlf)
  (?*assessmentReport* addInconsistentArea ?sa ?avg))


;;
; Rule 5:
;;;;;;;;;;;;;

;;
; If a particular response on a particular survey is unexpected, then
; print to the screen, "Question X from Survey Y contains an unexpected
; outcome."
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;

(defrule question_is_anomaly_rule
  "The response to this question is unexpected"
  (doRule5)
  (outcome_is_anomaly
    (questionNumber ?q)
    (surveyNumber ?sn)
```

```
    (squadronNumber ?sqdrn))
  =>
  (printout t "Question "  ?q  " from Survey " ?sn  ", Squadron "
?sqdrn " contains an unexpected outcome." crlf))

(defrule interesting_question_anomalies_action
  (doRule5)
  (interesting_question_anomaly
    (questionNumber ?q)
    (frequency ?f))
  (question
    (number ?q)
    (text ?txt))
  =>
  (printout t "Question " ?q " has " ?f " unexpected outcomes." crlf)
  (?*integrityReport* addQuestionAnomaly ?q ?txt ?f))

(defrule interesting_respondent_anomalies_action
  (doRule5)
  (interesting_respondent_anomaly
    (surveyNumber ?svy)
    (squadronNumber ?sqdrn)
    (frequency ?f))
  =>
  (printout t "Survey " ?svy ", Squadron " ?sqdrn " contains " ?f "
unexpected outcomes." crlf)
  (?*integrityReport* addRespondentAnomaly ?sqdrn ?svy ?f))

;;
; Rule 6: Negative Question Assessment
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;
; If a question's average weighted answer is less than zero, then print
; to the screen, "Question X has a negative assessment."
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;

(defrule negative_question_assessment
  "The question has a negative weighted average"
  (doRule6)
  (question
    (number ?n)
    (avgWtdAnswer ?a)
    (text ?txt))
  (test (< ?a 0))
  =>
  (printout t "Question " ?n ", " ?txt ", has a negative assessment."
crlf)
  (?*assessmentReport* addWeakQuestion ?n ?txt ?a))


;;
; Actions
;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defrule display_assessment_report
  (declare (salience -100))
  (display_assessment_report)
  =>
  (call (fetch agentApp) display ?*assessmentReport*))

(defrule email_assessment_report
  (declare (salience -100))
  (email_assessment_report (to $?toAddress))
  =>
  (call (fetch agentApp) mail ?*assessmentReport* $?toAddress))

(defrule print_assessment_report
  (declare (salience -100))
  (print_assessment_report)
  =>
  (call (fetch agentApp) print ?*assessmentReport*))

(defrule display_integrity_report
  (declare (salience -100))
  (display_integrity_report)
  =>
  (call (fetch agentApp) display ?*integrityReport*))

(defrule email_integrity_report
  (declare (salience -100))
  (email_integrity_report (to $?toAddress))
  =>
  (call (fetch agentApp) mail ?*integrityReport* $?toAddress))

(defrule print_integrity_report
  (declare (salience -100))
  (print_integrity_report)
  =>
  (call (fetch agentApp) print ?*integrityReport*))

(defrule save_assessment_report
  (declare (salience -100))
  (save_assessment_report)
  =>
  (call (fetch agentApp) save ?*assessmentReport*))

(defrule save_integrity_report
  (declare (salience -100))
  (save_integrity_report)
  =>
  (call (fetch agentApp) save ?*integrityReport*))
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. DATABASE SQL DDL

```
--
-- BuildTables.sql
-- Jan. 31, 2000
-- Description: DDL for building Survey3_DB database tables.
-- Rusl Flowers
--

CREATE TABLE Communities
(Community VARCHAR(25),
 CONSTRAINT Communities_PK PRIMARY KEY (Community))

CREATE TABLE Locations
(Location VARCHAR(25),
 CONSTRAINT Locations_PK PRIMARY KEY (Location))

CREATE TABLE Squadrons
(Squadron_ID INT,
 Community    VARCHAR(25) NOT NULL,
 Location     VARCHAR(25) NOT NULL,
 CONSTRAINT Squadrons_PK PRIMARY KEY(Squadron_ID),
 CONSTRAINT Community_FK FOREIGN KEY(Community) REFERENCES Communities,
 CONSTRAINT Location_FK FOREIGN KEY(Location) REFERENCES Locations)

CREATE TABLE Squadrons_Survey_IDs
(Survey_ID   INT,
 Squadron_ID INT,
 CONSTRAINT Squadrons_Surveys_IDs_PK PRIMARY KEY
(Survey_ID,Squadron_ID),
 CONSTRAINT Squadron_ID_FK FOREIGN KEY (Squadron_ID) REFERENCES
Squadrons)

CREATE TABLE Ranks
(Rank VARCHAR(25),
 CONSTRAINT Ranks_PK PRIMARY KEY (Rank))

CREATE TABLE Designators
(Designator VARCHAR(25),
 CONSTRAINT Designators_PK PRIMARY KEY (Designator))

CREATE TABLE Status_Values
(Status VARCHAR(25),
 CONSTRAINT Status_Values_PK PRIMARY KEY (Status))

CREATE TABLE Services
(Service VARCHAR(25),
 CONSTRAINT Services_PK PRIMARY KEY (Service))

CREATE TABLE Aircraft_Types
(Aircraft_Type VARCHAR(25),
 CONSTRAINT Aircraft_Types_PK PRIMARY KEY (Aircraft_Type))
```

```
CREATE TABLE Respondents
(Survey_ID INT,
 Squadron_ID INT,
 Survey_Date DATE NOT NULL,
 Rank VARCHAR(25) NOT NULL,
 Designator VARCHAR(25) NOT NULL,
 Flight_Hours INT NOT NULL,
 Type_Hours INT NOT NULL,
 Dept_Head BIT NOT NULL,
 Status VARCHAR(25) NOT NULL,
 Service VARCHAR(25) NOT NULL,
 Aircraft_Type VARCHAR(25) NOT NULL,
 CONSTRAINT Respondents_PK PRIMARY KEY (Survey_ID, Squadron_ID),
 CONSTRAINT Resp_Survey_ID_FK FOREIGN KEY (Survey_ID, Squadron_ID)
REFERENCES Squadrons_Survey_IDs,
 CONSTRAINT Rank_FK FOREIGN KEY (Rank) REFERENCES Ranks,
 CONSTRAINT Designator_FK FOREIGN KEY (Designator) REFERENCES
Designators,
 CONSTRAINT Status_FK FOREIGN KEY (Status) REFERENCES Status_Values,
 CONSTRAINT Service_FK FOREIGN KEY (Service) REFERENCES Services,
 CONSTRAINT Aircraft_Type_FK FOREIGN KEY (Aircraft_Type) REFERENCES
Aircraft_Types)

CREATE TABLE Safety_Areas
(Safety_Area VARCHAR(50),
 CONSTRAINT Safety_Areas_PK PRIMARY KEY (Safety_Area))

CREATE TABLE Questions
(Question_ID INT,
 Question_Text VARCHAR(255) NOT NULL,
 Is_Positive BIT NOT NULL,
 Safety_Area VARCHAR(50) NOT NULL,
 CONSTRAINT Questions_PK PRIMARY KEY (Question_ID),
 CONSTRAINT Safety_Area_FK FOREIGN KEY (Safety_Area) REFERENCES
Safety_Areas)

CREATE TABLE Answers
(Survey_ID INT,
 Squadron_ID INT,
 Question_ID INT,
 Answer INT NOT NULL,
 CONSTRAINT Answers_PK PRIMARY KEY (Survey_ID, Squadron_ID,
Question_ID),
 CONSTRAINT Ans_Survey_ID_FK FOREIGN KEY (Survey_ID, Squadron_ID)
REFERENCES Respondents,
 CONSTRAINT Ans_Question_ID_FK FOREIGN KEY (Question_ID) REFERENCES
Questions)

CREATE TABLE Questionnaires
(Questionnaire_Version INT,
 Last_Update DATE NOT NULL,
 CONSTRAINT Questionnaires_PK PRIMARY KEY (Questionnaire_Version))

CREATE TABLE Questions_Questionnaires
```

```
(Questionnaire_Version INT,
 Question_ID INT NOT NULL,
 Question_Number INT NOT NULL,
 CONSTRAINT Questions_Questionnaires_PK PRIMARY KEY
(Questionnaire_Version, Question_ID),
 CONSTRAINT QQ_Question_ID_FK FOREIGN KEY (Question_ID) REFERENCES
Questions,
 CONSTRAINT QQ_Questionnaire_Version_FK FOREIGN KEY
(Questionnaire_Version) REFERENCES Questionnaires)

-- end BuildTables.sql
```

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Rd., Ste. 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, California 93943-5101

3. Chairman, Code CS.....1
   Naval Postgraduate School
   411 Dyer Road
   Monterey, California 93943-5100

4. Dr. C. Thomas Wu, Code CS/KA.....2
   Naval Postgraduate School
   Monterey, California 93943-5100

5. Dr. A. Ciavarelli.....2
   The School of Aviation Safety
   Naval Postgraduate School (Code 10)
   1588 Cunningham Rd., Rm 301
   Monterey, California 93943-5202

6. Mr. Richard Healing.....1
   Department of the Navy Safety and Survivability
   Bldg. 36, Washington Navy Yard
   901 M Street, S. E.
   Washington, D.C. 20374-5028

7. CPT Thomas R. Flowers.....2
   214 Remagen Rd.
   Seaside, California 93955

8. LT David Dowler.....2
   313 Davis St.
   Portsmouth, Rhode Island 02871