NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

DATA AND MODEL MANAGEMENT FOR THE JOINT WARFARE EXPERIMENTAL PROTOTYPE

by

Michael T. L. Chua

September, 1996

Principal Advisor:	Mark A. Youngren
Associate Advisor:	Hemant Bhargava

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 September 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE DATA AND MODEL MANAGEMENT FOR THE JOINT WARFARE EXPERIMENTAL PROTOTYPE

5. FUNDING NUMBERS

6. AUTHOR(S) Michael T. L. Chua

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis describes a new data management design for the *Joint Warfare Analysis Experimental Prototype (JWAEP)*, a joint theater level, low resolution stochastic simulation developed at the Naval Postgraduate School. The design calls for (1) a 32 bit Windows program to access the JWAEP data which is stored in a SQL database server, (2) direct output of the database information into the plain text input files on the Unix host machine, and (3) remote execution of the JWAEP model via the network.

The viability of this design is demonstrated in the JWAEP Management Information System (JMIS) prototype program. JMIS is shown to be capable of achieving the stated design features; however, due to the size of the JWAEP database, it has not provide a full implementation of each of the design features. In addition, this thesis discusses the issues that have to be considered to maintain JMIS in synchronization with future developments in JWAEP.

14. SUBJECT TERMS Database, Graphical User Interface, Theater Level Simulations			15. NUMBER OF PAGES 91
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

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.

DATA AND MODEL MANAGEMENT FOR THE JOINT WARFARE EXPERIMENTAL PROTOTYPE

Michael T. L. Chua
Major, Singapore Army
BSC(ECONS), London School of Economics, 1989

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1996

Author: _____
Michael T. L. Chua

Approved by: _____
Mark A. Youngren, Principal Advisor

_____
Hemant Bhargava, Associate Advisor

_____
Frank C. Petho, Chairman
Department of Operations Research

ABSTRACT

This thesis describes a new data management design for the *Joint Warfare Analysis Experimental Prototype (JWAEP)*, a joint theater level, low resolution stochastic simulation developed at the Naval Postgraduate School. The design calls for (1) a 32 bit Windows program to access the JWAEP data which is stored in a SQL database server, (2) direct output of the database information into the plain text input files on the Unix host machine, and (3) remote execution of the JWAEP model via the network.

The viability of this design is demonstrated in the JWAEP Management Information System (JMIS) prototype program. JMIS is shown to be capable of achieving the stated design features; however, due to the size of the JWAEP database, it has not provided a full implementation of each of the design features. In addition, this thesis discusses the issues that have to be considered to maintain JMIS in synchronization with future developments in JWAEP.

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

This thesis proposes a new data management design for the *Joint Warfare Experimental Prototype (JWAEP)* model. A new design is required as the current input system, consisting of plain text files, lacks significant operating capabilities that are essential in a production combat model.

The proposed design calls for the development of a 32 bit Windows based program as a front end for the user to insert and modify JWAEP scenario data. This windows program will (1) interface with the SQL based database server, (2) output the required data files directly to the Unix host machine, and (3) activate the JWAEP model remotely via the network.

The viability of the proposed design is demonstrated in the JWAEP Management Information System (JMIS) prototype program. JMIS incorporates all the design features stated above, however, due to the size of JWAEP database, it has only achieved partial implementation of these design features.

This thesis also looked at the longer term issues that have to be considered when additional features are incorporated into the JWAEP model. It discusses the steps needed to keep both programs in sync and also ways to allow JMIS to be used with newer versions of JWAEP.

I. INTRODUCTION

A. OVERVIEW

The purpose of this thesis is to describe a new data management design for the *Joint Warfare Analysis Experimental Prototype (JWAEP)*, a joint theater-level, aggregated, low resolution simulation developed at the Naval Postgraduate School (NPS). The current data management system consists of multiple plain text files organized primarily by algorithm modules and has little regard to the logical structure of the data entities.

The proposed data management system incorporates a relational data model, a graphical user interface and integrates them into JWAEP without requiring substantial revision to the existing input routines. Chapter II is a review of the data management design of similar combat models. Chapter III provides an overview of the proposed design and also highlights the features that have been implemented in the prototype database management program. Chapter IV describes the procedures to maintain the usability of the data management program as further changes to the model are implemented. A summary of the thesis and ideas for enhancements to the data management system are presented in Chapter V.

B. BACKGROUND[Ref 1]

The ***Joint Warfare Analysis Experimental Prototype (JWAEP)*** is an interactive, two-sided, theater level combat model based on an arc-node representation of ground, air and littoral combat. It can be run in an interactive gaming mode or a closed-form stochastic analysis mode. The level of detail used in JWAEP is appropriate to represent battalion to brigade sized maneuver units, flight groups, and major combatant vessels. JWAEP is a software prototype developed by the Naval Postgraduate School for research and experimentation in command, control, communications, and intelligence (C^3I) centered approaches to modeling theater-level combat.

Ground warfare is executed upon the arc-node representation of the key terrain, objectives, defensive points and maneuver corridors. Units have the ability to move through

the network according to appropriate movement rates and terrain restrictions that are based on the size and maneuver capabilities of each unit. Attrition is assessed through the COSAGE/ATCAL process developed at the U.S. Army Concepts Analysis Agency.*

Air warfare is executed on a separate air grid. The air space within a theater of operations is divided into a user-defined grid, the air grid. Each grid square represents the volume of air from the ground up within the geographic area enclosed by the square. Air-to-air engagements are fought when aircraft encounter each other in a grid square; surface to air and air to surface engagements are fought between flights within an air grid and any ground targets or weapon systems on the terrain underlying the grid. The air grid is represented in the model as a node with direct connectivity to the eight adjacent nodes. Given the air structure, aircraft can choose a "least cost" path through the network to move to an engagement/target area and return. The air-to-air and air-to-surface engagements are adjudicated using the attrition mechanisms in the Air Forces Studies and Analysis Activity's THUNDER model. Surface-to-air engagements are adjudicated using a high resolution algorithm developed at NPS based on THUNDER algorithms. The littoral warfare module is currently under development. (Youngren and Lovell, 1996).

Data input manipulation is a common problem with computerized simulations of combat. This is due to (1) the size of the programs, (2) the constant revisions and updates of the models, (3) the uncertainty associated with basic input parameters, and (4) the requirement for scenario related data(as well as physical data), which tends to be hard to visualize in time and space. Chapter I I of this thesis will provide a brief survey of related combat models and their approach to data management.

---

* *COSAGE (Combat Sample Generator):* A stochastic division level model used to generate engagement input data for use in ATCAL. The input data is generated based on a specific area of operations and opposing forces. *ATCAL (An Attrition Model Using Calibrated Parameters)*: An attrition model in which calculations are made using high resolution results - results of a simulation resolved down to the interactions between individual weapons. It provides a loss-by-cause table (killer victim scoreboard), allocation of fire among all shooter and target types, expenditures of ammunition, and the relative importance of all weapons. [US Army Concepts Analysis Agency, CAA-TP-83-3, August 1983]

C. THE PROBLEM

The data input system in JWAEP is designed to facilitate the development of algorithms and code modules for the model. Input data files were created whenever new features and algorithms were added. This approach to data input is fairly typical of simulation model development in general and is illustrated in Hume's thesis [Ref 1] where he proposed the addition of a number of data files for the implementation of the attack helicopter module in JWAEP. One problem with this approach is that data concerning a single entity could be stored in many separate data files. For example, information on sensors can be found in various files such as *sensor.dat*, *sensorschedule.dat* and *sensorprob.dat*. While this arrangement of input files may make sense in terms of the organization of the code modules, it does not agree with the way a typical user would view the data, i.e., the sensor object.

The data required for running JWAEP, version 2.1, are stored in thirty-six plain text files. Some of these files contain information that is not referenced elsewhere, i.e., local information, while others contain information that is referenced externally, e.g., *weapon.dat, equipment.dat, aircraft.dat, etc.* There are no built in mechanisms that ensure that a *weapon id* referenced in the *leathalarea.dat* actually exists in the *weapon.dat* file. While such referential integrity constraints are standard features in databases, it is much harder to implement in simulation models without substantial amount of coding.

Beside the issue of referential integrity, the plain text data file does not support automatic type checking. Therefore it is possible for the user to put values into the data fields that are not within the permissible ranges. Similarly, this is not a problem in databases as the server performs type checking when the data is keyed into the system. The model currently performs some basic type checking in its input routines. These inbuilt type checking routines are able to identify some of the data integrity problems mentioned but they become increasingly ineffective as the amount of information in the database grows. More and more codes have to be added to the routines as additional data structures are added to the model. If such routines are not added, then the possibility of data errors would increase. On the other hand, the addition of such codes will adversely impact the run time of the model. Furthermore, if we consider the process of data entry into the model, it would be obvious that

type checking should be a one-time event performed as new information is keyed into the database. A design that consistently performs type checking as a routine part of the model function adds considerably to the run time without significant additional benefit to data reliability. Thus it is more efficient to have an elaborate type checking routine that is triggered during the data entry process. Such an elaborate type checking routine will not affect performance as the routine examines a small data set each time.

Another problem associated with referencing data variables from one input file to another is the difficulty of identifying the data element that has been indirectly referenced. For example, the *unit.dat* file contains a data type call *unit type* which contains multiple elements of the *weapon type*. In the *unit.dat* file, these *weapon type* are only referenced by the *weapon type id*. Unless the user already memorizes the *weapon type ids*, the *id* number would make very little sense to him. This type of problems is typical in databases where each normalized table contains information about a single object or relation and little else. However, in the realm of database technology, this is not a problem as the user is presented a virtual view of the data, i.e., a composite image of the information that is derived from multiple tables. Therefore relevant information such as the *weapon description* can be presented together with the *weapon id* even though that piece of information does not belong physically to the table being edited.

The present data structure is designed for the input of the data from a single scenario. This design creates a lot of difficulties when we need to use the model to examine multiple scenarios. In the best case where the multiple scenarios are similar, e.g., excursions from a base scenario, the process may simply be a mass copying of the data files and minor editing work. However, when we wish to undertake a new analysis in a different theater, it will require a wholesale revision of the data files. Either way, the analyst will have to manually copy the required files and then edit the relevant sections to reflect the desired changes. Such a manual process will, in most likelihood, introduce errors into the data and necessitate careful debugging.

Another problem associated with having to create multiple sets of data files for different scenarios is that of data consistency. Any changes to the data variables in one

scenario would not be reflected in other scenarios that make use of that variable. Therefore, the onus is on the analyst to edit the data field in each and every scenario data file that is affected. If the analyst is not thorough, i.e., only modifies "current" scenarios, a situation may arise where another analyst using an "older" scenario may be using invalid data.

The use of a text editor is not an effective means for entering some of the required data elements in JWAEP. Data structures such as *nodes*, *arcs* and *units* have geographical attributes which are better suited for entry and editing via a graphical interface. A graphical interface is more intuitive and efficient as it is easier to "drag" a unit from one node to another than to edit a text file to modify the latitude and longitude fields. With a graphical interface, the user will be able to visualize the changes immediately, whereas the current system requires the model to read in the modified information before displaying the changes on screen.

D. THE PROPOSED SOLUTION

The proposed data management system comprises a client personal computer (PC) running Windows 95© or NT© accessing the model's database using a windows based graphical user interface. The database could either reside locally on the hard disk or could be residing on a Standard Query Language (SQL) server elsewhere on the network. The client PC would then be able to activate the JWAEP model through remote procedure call and have it displayed locally through the X window server. Chapter III provides a complete description of the proposed design.

Borland's Delphi© Developer, Version 2, is chosen as the tool for developing the windows front end. The accompanying local Interbase Server and its suite of SQL tools are used as the SQL server and for the development of the data model respectively. "Sockets," a freeware Delphi component developed by Gary T. Desrosiers, is used for remote connectivity between the unix host and the PC client. "TWorldmap," a commercial delphi component developed by Don Bauer, is used to create the mapping capability for the program. "Samba", a freeware Unix Session Message Block (SMB) server developed by Andrew Tridgell, is used to provide file mounting services on the Unix workstation.

II. REVIEW OF OTHER COMBAT MODELS

This chapter examines briefly some of the production combat models currently in used for wargaming and analyses in US defense establishment. The purpose of the review is to identify the database design that is incorporated in each of the model and to learn the desirable features that should be included in the proposed data management system. The combat models are chosen based on the scope of the model, initial knowledge of its database design, its widespread use in military establishments, and also more importantly the access to the model for the review.

A. JANUS

JANUS is a multipurpose ground combat simulation wargame. It is an interactive, near-real-time model developed to explore the relationships of combat and tactical processes. Players make doctrinal and tactical decisions, deploy forces, develop scenarios, and make and execute plans. It also serve as a Battle Focus Trainer to assist Commanders at Battalion level and below in training subordinate leaders in decision making processes[Ref 2].

The model is widely used in US Army establishments, e.g., TRAC-WSMR, TRAC-FLVN, Ft. Benning, Ft. Knox, Ft. Rucker, Ft. Sill, Ft. Lee, etc., and other allied countries such as Australia, France, Germany, and the United Kingdom. The latest versions for the VMS and HP Unix platforms are v5.8 and v6.0 respectively. This JANUS review is based on limited hands-on experience using the HP Unix version 6.0 of the model and information gleaned from the JANUS 3.X User Manual and Data Base Manager's Manual[Ref 3, 4].

1. Database Overview

JANUS uses a hierarchical database system for the storage of its plain text data files. This means that the data are stored in plain text files under various sub-directories that are grouped according to functional areas. Access to the bulk of the database is provided via a text based menu system, although a significant portion of the data editing is performed via a graphical user interface.

2. Multiple Menu Layers to Data

Much of the actual data fields are nestled deep in the menu system, i.e., they are only accessible through the sub-sub-menu of the sub-menu of the main menu. Traversal from one data type to the next would often require backing up to the root of the menu system before descending to the desired data fields.

3. Scenario Verification Program

As there is no external database server managing the data, the model essentially has to use its own database system to manage the data. This system uses a scenario verification program to detect anomalies in the database prior to model execution. Therefore the user is "free" to enter any type of data during the editing process and will only be able to determine data errors when the verification program is used. This is probably why the model comes with a master database, with properly verified data, and a developmental database, for testing new data elements and fields.

4. Graphical Interface

The model has a very good interface implementation for handling data elements that have geographical attributes, e.g., unit routes of advance, engineering obstacles, etc. Much of the data entry process for such elements uses the point and click interface. However, the implementation of the graphical interface is incomplete as the user is unable to add units into the scenario using the GUI and has to revert to the menu system.

B. JOINT CONFLICT MODEL (JCM)

JCM is a distant cousin to the JANUS model. It was developed in 1991 as a part of Pacific Command's Joint Training Forces Simulation System (JTFSS) and is also used as an exercise driver for training of Joint Task Force commanders and staffs. It has the ability to represent Joint and combined conventional and unconventional forces with up to 5 sides. The model can represent either individual platforms (trucks, tanks, helicopters), individual soldiers, or aggregated heterogeneous units up to Brigade size. Attrition is adjudicated through a stochastic process involving acquisition and probability of hit and probability of kill given hit[Ref 5].

Current JCM users include PACOM, USSOCOM, Marine Corps Wargaming Center, USACOM, USSOUTHCOM, US Army, JRTC, and SOF SIM Center. The review of the model's database design and implementation is based entirely on the information in the JCM Simulation Manual version 2.4 and Scenario Editor Manual version 2.4[Ref 6,7].

1. Database Overview

The model uses a similar design to JANUS for its database. The data is essentially organized in files and directories. Users access the database via text based menu options. The model lacks a dedicated database server back end, thus all data verification is performed internally through the data editors.

2. Multiple Editors

The model uses the scenario editor and a PH/PK editor to modify the text based data entries, the terrain editor for terrain information, and JCM itself for *planning data*, i.e., data elements with geographical attributes.

3. Multiple Layers to Data

The model has a tree based text menu system for access to the various data editors. Thus the user has to traverse through the different menu options to get to the desired data entries.

4. Graphical Interface

The GUI for editing data elements with geographical attributes is more complete than JANUS. The program can add units to the scenario directly via the GUI rather than the menu system. This is more efficient than using the menu system to create units and then using the model to position the units to the correct locations.

C. JOINT THEATER LEVEL SIMULATION (JTLS)

The Joint Theater Level Simulation (JTLS) system is an interactive multi-sided analytical tool that models a joint air, land, and naval warfare environment. It is designed as a theater-level model for use in the following areas:

a. The analysis, development, and evaluation of contingency plans and joint tactics,

b. The evaluation of alternative military strategies, and

c. The analysis of combat systems.

The model is also useful as a situation driver and combat evaluation tool for joint and international staff exercises. [Ref 8]. Current users include USACOM, USCENTCOM, USEUCOM, USSOCOM, USSOUTHCOM, Joint Warfighting Center, SHAPE Technical Center, WPC, NDU, AUCADRE, Army War College, Naval Postgraduate School and Combined Forces Command/Korea. The review for this model is based primarily on the model documentation though there was limited exposure to the actual program.

1. Database Overview

JTLS has a more sophisticated database management design which uses a dedicated INGRES relational database system. The data used in JTLS are described in terms of relational tables and the interrelationship of these tables.

2. Database Editing

There are two main programs used in JTLS, the Scenario Development System (SDS) and the Scenario Preparation Program (SPP), for database editing. The SDS is used primarily for the creation of new databases and for the updating of existing scenarios that require substantial changes. The SPP is used primarily for making small changes to the current scenario.

3. Text Files

The SDS and SPP create plain text scenario files that are used by the Combat Events Program(CEP). The CEP is essentially the system running the JTLS simulation. JTLS also comes with a scenario verification program to ensure the consistency and integrity of the plain text files.

4. Graphical Interface

The current version of JTLS also does not have a complete implementation for the graphical editing of unit data. The SPP is still required to define the combat elements in the scenario and only then can subsequent modifications be made using the Model Interface Program, i.e., the player's console.

5. Advanced Help System

Each data screen has an associated help screen that explains the purpose and operation of that screen. In addition, JTLS has a Manual Generation Program which generates text files of the database elements used in the scenario for player review.

6. Current Development.

Rolands and Associates, the model's developer, is currently developing an upgrade to JTLS. The upgraded version will be based on a ORACLE database server and has a GUI that is capable of inserting units into the scenario.

D. COMMON DESIGN FEATURES

The following summarizes some of the common design elements in these combat models.

1. Multiple Stage Processing

Data entry is usually handled by separate software programs and frequently more than one program is used. Thus the main model is not burdened with the tasks of ensuring the integrity of the database and can devote the run time to model execution. The number of stages used depends on the complexity of the database as well as the sophistication of the model.

2. Strong Graphical Interface

While the complete functionality of the graphical interface is not present in all the models, those available are highly sophisticated. The user is able to perform most of the required tasks (editing the unit's deployment and movement) via the GUI. In addition, the user is also able to query the graphical elements to determine its detailed contents.

3. Single Platform But Multiple Computer Systems

The models and the data editors can be executed directly from a single computer whether its a VAX or a Unix machine. However, the design of these models is such that more than one computer system can be used at the same time. Different computers are used to send orders to the model or to display the run time graphics for the different sides in the scenario.

E. SUMMARY

The emphasis of a good graphical interface for the display of the combat model's entities should be emulated. However, the hierarchical system for accessing other system data elements seem rather cumbersome and dated. The use of a specialized database backend is not common though the need for database type functionality is universal judging from the frequent documentation warnings about data validity, consistency, etc.

III. PROPOSED JWAEP DATA MANAGEMENT SYSTEM

A. OVERVIEW

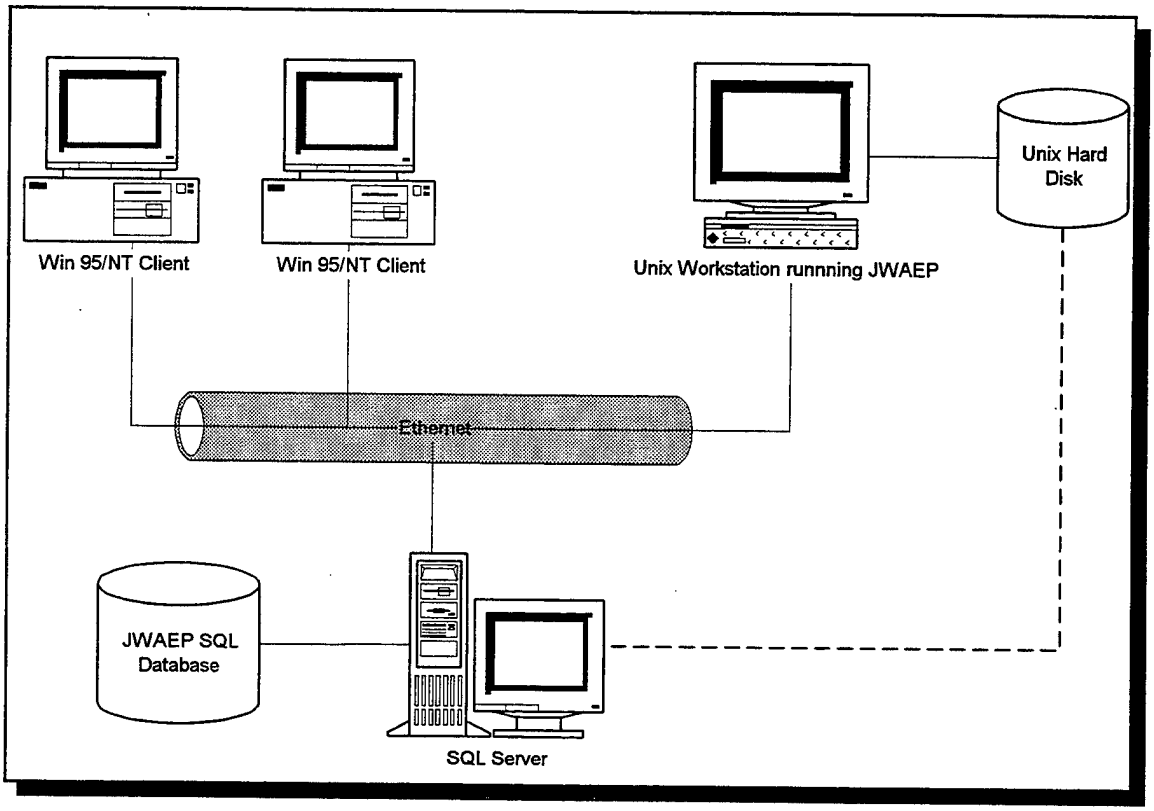

Figure 1 Proposed design of JWAEP data management system

Figure 1 gives an overall view of the proposed JWAEP data management system. The three components in the system are the existing JWAEP model, a SQL database server and PCs to access the database and to activate the workstation to execute the model. This design does not require any changes to the input routines in JWAEP except for possible streamlining of the codes to take advantage of the type checking functions performed by the database. It is also a scalable design as we can collapse the SQL database and the PC client access to a single PC system for a minimal configuration. At the same time, we have the flexibility to add more PCs and workstations as well as a dedicated SQL server should there be a need for concurrent analysis or greater computation power.

In the next section, the data model underlying the proposed system will be described. This will be followed by a detailed description of components of the system as presented in

the overview. Subsequently, the status of the prototype program will be briefed and the completed prototype program functionality presented.

B. DATA MODEL

1. Overview

We need to develop the underlying data model before we can consider the design of the user interface. This stage is crucial as mistakes made here will have a cascading effect on subsequent developments. Rectification of errors in the data structure during the user interface development will entail creating temporary tables to hold existing data, dropping integrity constraints or tables, recreating the modified tables and then moving the data from temporary storage back to the newly created tables. This is a painstaking process and its frequency of occurrence is very dependent on the care taken in the creation of the data model. This important lesson was brought home as the process was repeated a number of times during the development phase.

2. Table Creation

The creation of the data model starts with the detailed analysis of the input files to derive the tables and relations that exist in each file. Typically, most data files contain a few data structures and local references that are used by the specific algorithms. Such localized data are relatively easier to convert as they translate directly into standard database tables and integrity constraints. However, there are also instances where the information for a logical data structure is spread out in several of the data files. An example is the data for *sides* in the scenario, where information has to be culled from files such as *side.dat*, *grdrules.dat*, *airrules.dat*, *unit.dat*, *sitrep.dat* and *sensor.dat*. In such cases, there is a need to examine carefully whether such diverse data should be grouped logically into a single structure. If that is so, it is essential to document the original source location of the data structure sub components. This will facilitate description of these composite tables during the documentation process.

3. Eliminating Redundant Data

The existing data files also contain instances of redundant or replicated information. For example, the data field *side*, used to indicate ownership of a particular unit, is present in the data structure for *unit group, unit type* and *unit*. Since *unit type* has a data field indicating its *unit group* it would not be necessary to define a separate *side* field and in fact may create the potential for conflict when the data in each of the *side* field do not agree. This same argument also holds for *unit* which is a specific instance of a *unit type*. Therefore the tables in the proposed data model do not parallel exactly those in the original input as all duplicate data fields were eliminated.

4. Separation of System and Scenario Data

The design of the data model has to provide for the separation of the data structures into system and scenario dependent data. System data are those data structures that do not change from one scenario to the next, e.g., *aircraft*, *equipment*, *sensors*, etc. Such basic structures would be used for building actual data instances in a scenario, e.g., *squadrons*, *units*, *unit orders*, etc. The separation of the data into two categories is essential when we want to use the model for multiple scenarios. The problem of data inconsistency would be lessened as each scenario would reference the same copy of system data. Changes to the system data need to be applied to a single point in the database against the case where all copies of the data have to be modified.

5. Non JWAEP Information

The existing data files do not contain information that are essential for a system used by multiple users running multiple scenarios. Therefore the supporting data structures for multiple scenarios and users have to be considered and developed as part of the overall data model. Information that are extraneous to the model input requirement includes scenario creation information, e.g., scenario owner, date created, modification date, etc. and system operation information, e.g., Unix host names, data directory location, client workstation address, etc. The use of such data will be discussed later in the next section.

6. Considerations for Multi-Sided Data

Another issue is that the model currently only supports the representation of two sides. Thus the *side* variable used in the model is limited to values 1 and 2. However, in a populated database, we will actually have system data from many different nations(*sides*). Thus the definition of *side* in the our database have to be different from that used in the model. Therefore, we will have to develop a mechanism to translate the *side* information prior to model execution. This approach will also serve a longer term need when the model is enhanced to include the ability to represent multi-sided warfare.

7. Completed Data Model

The following list is a summary of the tables created for the proposed data model and information relating to its source input file as well as external references. Detailed information on each of the table can be viewed in Appendix A.

Table	Description	Original Input File	External References
JUSER	JWAEP user	*New table*	NIL
SCN	Scenario	*Some fields from sensor.dat*	JUSER
EQUIPGP	Equipment group	*equipment.dat*	NIL
SYSSIDE	System side	*New table*	NIL
SIZ	Unit size	*icon.dat*	NIL
SENSOR	Sensor	*sensor.dat*	NIL
TERRAIN	Terrain	*class.dat*	NIL
WPN	Weapon	*weapon.dat*	NIL
RADAR	Radar	*radar.dat*	NIL
FCN	Unit function	*unit.dat*	NIL
ACTGTCLASS	Aircraft target class	*aircraft.dat*	NIL
ACMSNTYPE	Aircraft mission type	*aircraft.dat*	NIL
EWCLASS	Electronic warfare class	*ewdetect.dat*	NIL
ORDTYPE	Order type	*coa.dat*	NIL
NETSENSOR	Network sensor	*sensor.dat*	SENSOR
SCHSENSOR	Scheduled sensor	*sensor.dat*	SENSOR
EQUIP	Equipment	*equipment.dat*	EQUIPGP
SENSORSCHD	Sensor schedule	*sensorschedule.dat*	SCN, SENSOR
SENSORDETPROB	Sensor detection probability	*sensorprob.dat*	EQUIPGP, TERRAIN, SENSOR

Table	Description	Original Input File	External References
UNOPMOVRATE	Unopposed movement rate	*class.dat*	UNIT, TERRAIN
OPMOVRATE	Opposed movement rate	*class.dat*	SIZ, UNIT, TERRAIN
NODE	Node	*node.dat*	SENSOR
ARC	Arc	*arc.dat*	TERRAIN, SENSOR
SCNAVEAPPR	Scenario avenue of approach	*aveapproach.dat*	SCN
EQUIPDEN	Equipment density	*equipment.dat*	EQUIP, TERRAIN
SCNSIDES	Scenario sides	*parts from sensor.dat, side.dat grdrules.dat, airrules.dat, unit.dat, sitrep.dat*	SCN, SYSSIDE, STATFCN, SENSOR, UNITTYPE
GRAPHDISP	Graphic display	*graphics.dat*	SCN, JUSER
JAMMER	Jammer	*jammer.dat*	RADAR
SECADTYPE	Secondary air defense	*adtype.dat*	EQUIP, WPN
PRIADTYPE	Primary air defense	*adtype.dat*	RADAR, EQUIP, WPN
ADPRIANDSEC	Secondary air defense at primary air defense location	*adtype.dat*	PRIADTYPE, SECADTYPE
UNITGP	Unit group	*unit.dat*	SIZ, SYSSIDE, UNIT
UNITTYPE	Unit type	*unit.dat*	UNITGP, FCN, PRIADTYPE
UNITHIER	Unit hierarchy	*unit.dat*	UNITTYPE
UNITEQUIP	Unit equipment	*unit.dat*	UNITTYPE, EQUIP
SCNUNIT	Scenario units	*unit.dat*	SCN, UNITTYPE
AIRBASE	Airbase	*airbase.dat*	NODE, SYSSIDE
SCNAIRBASE	Scenario airbase	*new file*	SCN, AIRBASE
AIRCRAFT	Aircraft	*aircraft.dat*	NIL
ACEWDETPROB	Aircraft electronic warfare detection probability	*aircraft.dat*	AIRCRAFT, EWCLASS
ACMSN	Aircraft mission	*aircraft.dat*	AIRCRAFT, ACMSNTYPE, JAMMER
ACMSNEQUIP	Aircraft mission equipment	*aircraft.dat*	ACMSN, WPN
SQUADRON	Squadron	*squadron.dat*	AIRCRAFT
SCNSQUADRON	Scenario squadron	*squadron.dat*	SCNAIRBASE, SQUADRON
AIRAPPORT	Air apportionment	*airapportionment.dat*	SCNSIDES
STRATTGT	Strategic target	*strattgt.dat*	NODE, SCNSIDES

Table	Description	Original Input File	External References
SECADPK	Secondary air defense probability of kill	*sapk.dat*	SECADTYPE, ACTGTCLASS
PRICADPK	Primary air defense	*sapk.dat*	PRIADTYPE, ACTGTCLASS
ACDETECT	Aircraft detect probability	*aadet.dat*	AIRCRAFT
ACWPNCOMBO	Aircraft weapon combination	*aapk.dat*	AIRCRAFT, WPN
COMBOPK	Aircraft weapon combination probability of kill	*aapk.dat*	ACWPNCOMBO, ACTGTCLASS
COMBOADV	Aircraft weapon combination relative advantage	*aapk.dat*	ACWPNCOMBO
PTWPNPKEQUIP	Point weapon probability of kill on equipment	*pwpk.dat*	WPN, EQUIP
PTWPNPKRADAR	Point weapon probability of kill on radar	*pwpk.dat*	WPN, RADAR
LETHALAREAEQUIP	Weapons lethal are on equipment	*lethalarea.dat*	WPN, EQUIP
LETHALAREARADAR	Weapons lethal are on radar	*lethalarea.dat*	WPN, RADAR
FIREMSN	Fire mission	*firemsn.dat*	WPN, UNITTYPE
COA	Course of action	*coa.dat*	SCNSIDES, SCNAVEAPPR
COAUNIT	Units involved in the COA	*coa.dat*	COA, UNITTYPE, ORDTYPE
COAUNITNODE	Nodes and units involved in the COA	*coa.dat*	COA, UNITTYPE, NODE
ALTORD	Alternate orders	*alternord.dat*	COA, UNITTYPE, ORDTYPE
ALTCOAUNITNODE	Nodes and units involved in the alternate COA	*alternord.dat*	COA, UNITTYPE, NODE

B. DATA MANAGEMENT MODEL

1. JWAEP Model

One of the design goals for the data management system was that the proposed changes be transparent to the model. This means that there would not be a need to change

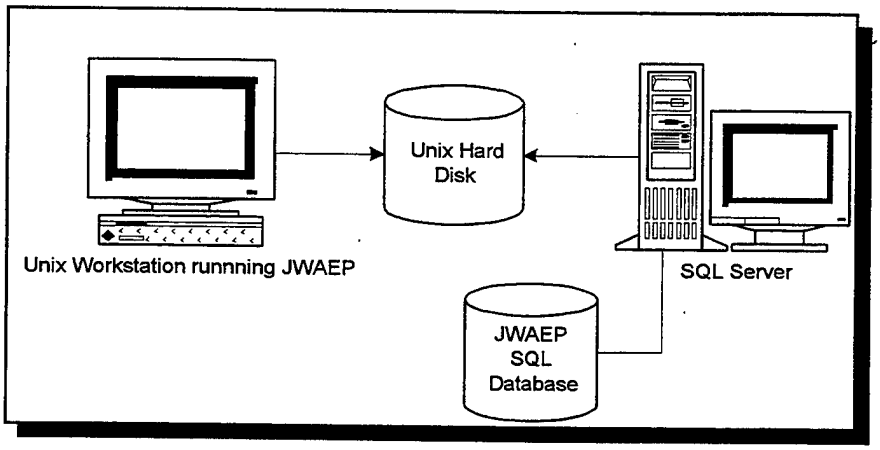

Figure 2 Data Input and Output for JWAEP Model.

any of the codes that are already implemented. This design goal is achieved as the data management system outputs the required data files into the Unix hard disk directly as shown in Figure 2. The JWAEP model would read the data files directly from the file system. There is no need for the model to communicate with the database server or to even be aware of its presence.

While the goal is to minimize changes to JWAEP, we should not refrain from streamlining the codes to take advantage of the new data management design when they exist. The first area that can be improved would be to eliminate the "comment stripping" routine that is called at the start of the input process. Comments in the input files are essential when the user edit the data directly as they help the user to understand the data structure and external references. Under the proposed design, the data files are generated by the database program and there would not be a need to add "comments" to the output. Therefore this function will become superfluous and should be eliminated when the data management system becomes functional.

The model currently contains within each input routine *read* statements that read in data fields labels which are then discarded. To ensure compatibility with the input routines, the output routine would have to include in the exact number of field labels into the output files. Since these labels are of no value to the model, its inclusion would be purely for the purpose of compatibility. A better approach would be to remove these *read* statements and thereby eliminate the need to print out the field labels. Unfortunately, the task of removing the read statements will require substantial editing of the source codes.

2. SQL Database

a. Scalability

The design for the data management system calls for a SQL based server to provide support for the data processing. This design is intended to allow the JWAEP data management system to be scaled easily according to the level of usage. Since SQL is an industry standard for databases, there is a wide choice of software packages that can be used. The type of SQL server used can vary from the standalone local Interbase server used for the development of the prototype to industry strength Unix based Oracle or Sybase SQL servers. The choice of an appropriate SQL server would depend on the expected load factor. Whatever the initial choice, we will still have the option to migrate the data in respond to usage growth as SQL data is easily ported from one server to another.

b. Concurrent Access

With a SQL server, it would be possible to have concurrent access to the database. Having the ability to have concurrent access means that more than one scenario can be worked on at the same time. This feature will come in handy when we have more than one analyst working on the same project or different analysts working on different projects. Common information can still be shared but at the same time each analyst can have their own copy of scenario data.

c. *Basic Type Checking*

Type checking is a standard feature of any database and the checking rules are defined when the data model is created. The type description of the fields in the *equip* table is shown below:

```
TABLE EQUIP
(EQUIPID VARCHAR(6) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
MODE VARCHAR(8) NOT NULL,
MXRANGE FLOAT NOT NULL,
PKTSIZE FLOAT NOT NULL,
MXERR FLOAT NOT NULL,
EQUIPGP VARCHAR(6) NOT NULL);
```

Basic type checking uses the field definition to determine whether the data entry is valid. For example the *equipid* field is defined as a variable 6 characters length field that must be filled; i.e., null value not accepted, when the record is created. This field is also defined as a primary key, meaning that each *equipid* be unique. The SQL server will use the type information to ensure that the correct type of data is entered. Invalid entries will result in an error message and will allow the user to rectify it.

d. *SQL Check*

Beside the basic type checking capability, SQL databases also offer the use of the CHECK command to provide an additional level of type checking automation. The SQL CHECK command can be very simple; e.g., ALTER TABLE EQUIP CHECK (MODE = "DIRECT" OR MODE = "INDIRECT"), which simply instructs the SQL server to send an error message when the user keys in data other than "DIRECT" or "INDIRECT," or it can be more complex with conditional statements with IF THEN ELSE logic. This capability will support the creation of elaborate type checking procedures that will be part of the data model and is independent of the user interface that accesses the database. Potentially it can also enhance performance as the processing is done at the server (assumed to be more capable) end than at the client end.

e. Efficient Copying of Scenario Data

The copying of scenario data from old to new scenarios can be automated easily using standard SQL commands. The following SQL statements illustrate the process of duplicating two new rows for the table *scnside* using a minimal list of SQL commands:

```
DELETE FROM TMPSCNSIDE;
INSERT INTO TMPSCNSIDE SELECT * FROM SCNSIDE
    WHERE SCN = "<SPECIFIC SCN ID>";
UPDATE TMPSCNSIDE SET SCN = "<NEW SCN ID>"
    WHERE SCN = "<SPECIFIC SCN ID>";
INSERT INTO SCNSIDE SELECT * FROM TMPSCNSIDE;
```

The above steps assumes that a *tmpscnside* exists and that it has the same structure as that of the original *scnside*. Relevant temporary storage tables can be created as part of the data model and distributed together with the database.

3. JWAEP Management Information System (JMIS)

a. Design Considerations

The initial design for JMIS was to host the program on the same Unix system as the JWAEP model. This arrangement is neater as there is only be one system to be configured which eliminates problems that would arise in a networked design. However, the Unix systems available in NPS OR department are standard Sun workstations lacking suitable application development packages. It would take a considerable amount of time and money to acquire the tools needed for the prototype development. As such, a PC based design was considered and evaluated for its suitability to the tasks required. In the evaluation, the main consideration was the increase in program complexity due to the need for communication between the PC and the workstation. But when this consideration is evaluated together with the requirement for concurrent access and scalability; i.e., the ability to add more computer systems in the system design, then the issue of increased complexity due to network communication becomes moot. The ultimate push towards a PC based design was the availability of suitable tools and software components that facilitate the development of a prototype system.

b. Main Features

JMIS is designed to be a 32 bit windows based program that would be used to access the database and to activate the JWAEP model. As JMIS is a 32 bit windows program, it can only be used on a 32 bit operating system such as Windows 95 or NT. The advantage of a 32 bit design is that the program does not have to be concerned about memory allocation, which is a major technical difficulty under 16 bit programming.

The program is designed to use the Window's Multiple Document Interface (MDI) standard which supports the use of multiple child windows within the main parent window. This design allows the user to open multiple windows to edit system and scenario data at the same time. The use of the MDI standard will help to facilitate the design aim of providing the user access to the underlying data within two mouse clicks; i.e., the ability to edit any data item without having to navigate through layers of screens and menu options.

A toolbar will also be incorporated as part of the user interface. The toolbar, popularized by Microsoft's suite of business applications, will allow users to have to easy access to important and frequently used program functions. Ideally, the toolbar functions should be configurable by the user to suit his personal preferences.

c. Interface with the SQL Server

Database programs developed using Borland's Delphi Developer use the Borland Database Engine (BDE) to configure the parameters necessary for data access. The standard BDE installation include drivers that can communicate to file based databases such as PARADOX© or DBASE© and server based system such as INTERBASE©. Additional drivers, e.g., SYBASE©, ORACLE© and MSSQL©, are included in Delphi's Client Server version of the development tool. Local communication to file based databases is quite straightforward, requiring only the appropriate data access drivers. Access to remote SQL servers is slightly more complex as it requires the program and server to communicate via the network. Fortunately, the BDE can communicate to the remote SQL servers via TCPIP, NETBEUI or IPX; i.e., all the major protocols used for remote communications. Thus it will be an easy task to configure the BDE to talk any SQL servers chosen for the production system.

d. Mounting the Unix File System

To execute JWAEP, we first need to translate the information from the database into the plain text files that the model requires. Secondly, we need to put the data files in a location where the model can access. There are three ways to move the data files to the appropriate locations in the Unix file system. The first is to *ftp* the files; i.e., use the File Transfer Protocol[Ref 13], from the client PC to the Unix workstation. The second is to mount the Unix file system via the Network File System (NFS) Protocol[Ref 14] and then output the data files to the PC's network drive as if it the drive is local to the PC. The third approach is to mount the Unix file system via the Session Message Block (SMB) Protocol[Ref 15] and use it as per the NFS approach. The first approach is more elaborate as it involves writing the output and then ftping the files to the other machine. However it has the advantage that no additional software would be required to be used on the client PC. The second and third approach are essentially the same as they require the client PC to run the NFS or the SMB client program. The difference for the client PC is that the NFS client software has to be purchased separately while the SMB client software is native to the Windows 95 and NT operating systems. At the server end, NFS is native to Unix systems while SMB has to be acquired from external sources. However there is a freeware SMB server program[Ref 9] available which has all the required functionality. As such, the third approach for mounting the Unix file system is chosen for our design. Theoretically, it would be possible to use more than one approach and let the user decide the approach to use in the specific implementation of the system.

e. Interface with the Unix Workstation

To execute the JWAEP program, JMIS must be able to send the execute command via the network to the Unix workstation. This can achieved via the Remote Execution *(rexec)* Protocol[Ref 17], where the commands are sent via TCPIP to the Unix

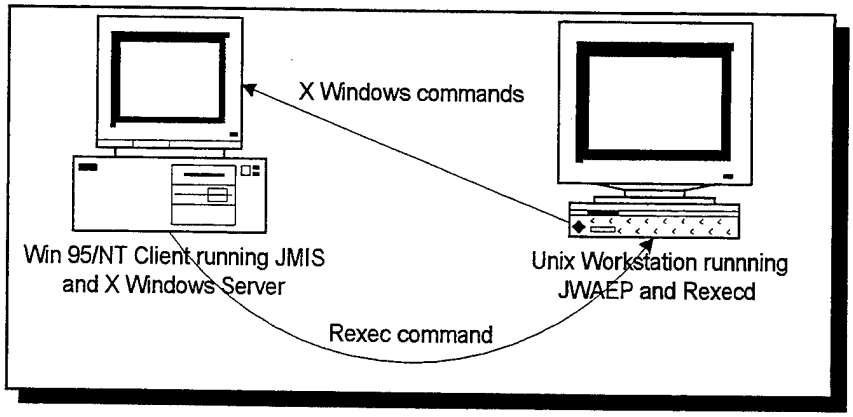

Figure 3 JMIS interactions with the Unix workstation.

workstation running the *rexec* daemon *(rexecd)*[Ref 18]. Once the user identity has been authenticated, the Unix machine will execute the command as requested. Beside the implementation of the *rexec* protocol, we need to run a X windows server so that JWAEP's runtime graphics will be displayed locally on the PC client. Figure 3 illustrates the relationship between JMIS and the workstation that has been outlined above. Additional details of the implementation are presented in the next section.

C. STATUS OF PROTOTYPE PROGRAM

1. General

The main design features mentioned in the previous section have been successfully tested although the complete functionality has not yet been implemented in JMIS. The level of implementation varies and is described in detail in the subsequent paragraphs. The testing and configuring of auxiliary software such as the SMB and X Windows server are also successfully completed.

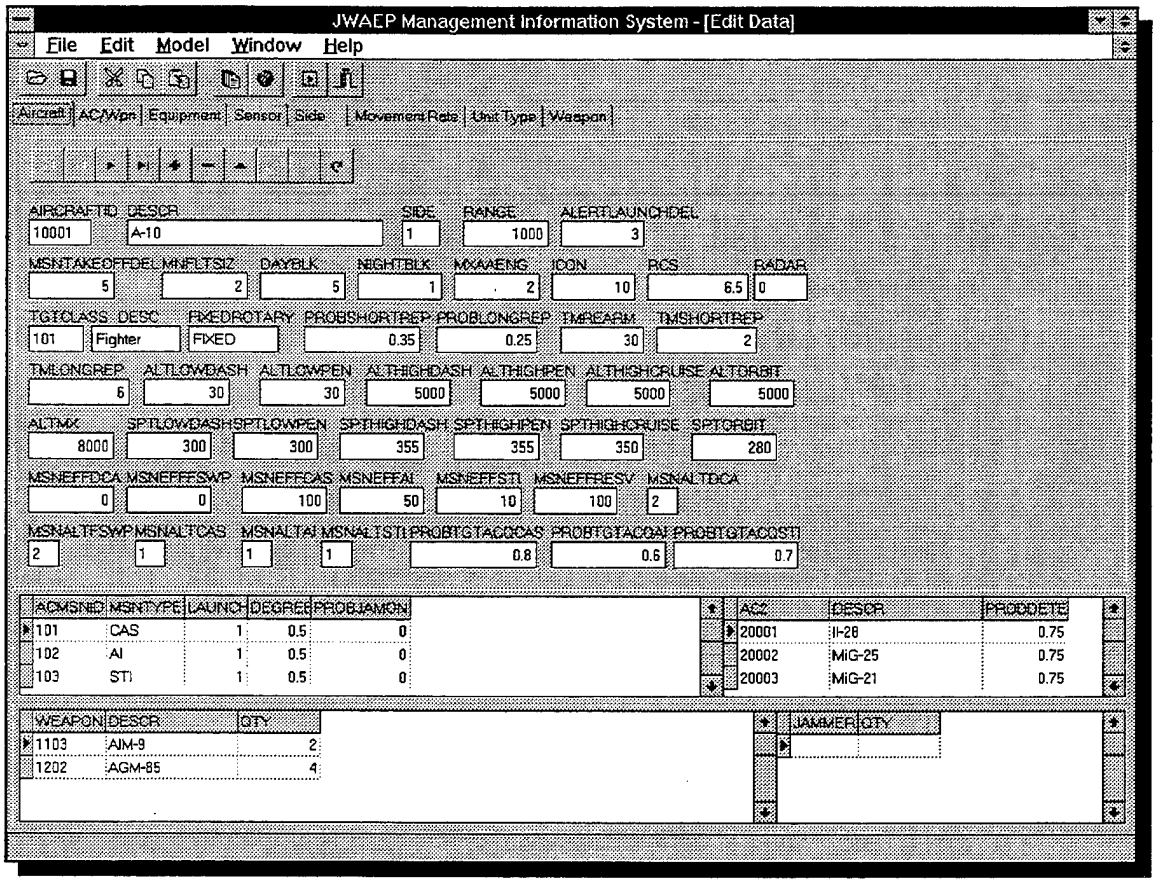

Figure 4 Main Program Screen with Edit Data Window Shown.

2. Main Program

Figure 4 shows the main program screen which also contains the *Edit Data* window. The user accesses the program functions through the menu selections shown at the top of the screen or via the toolbar which is positioned below the menu options. There are two main windows for the user to edit the JWAEP database. The first window, *edit data*, allows users to edit plain text database entries. The second window, *edit map data*, allows user to edit data that contains geographical attributes or are linked directly to data containing geographical attributes. The other two major functions accessible from the main window include the output selection and the JWAEP execution functions. The capabilities implemented in these functions will be presented next.

3. Edit Data

a. General

This part of the program is the most fully developed part of the JMIS prototype as it already implements access to twenty three of JWAEP tables. However, it is also the area where much additional work is required. The primary difficulty in the development of the prototype lies in the size of the overall database. Though the task for creating the links is relatively straightforward, it still requires considerable time to implement each data page. As such, only partial functionality is achieved.

b. Tabbed Sheets

The *edit data* window uses the tabbed folder metaphor to facilitate access to the different data entities. Related data entities are stored in sheets which are accessed by selecting the sheet tab entry. For example, Figure 4 shows the *edit data* window with the *Aircraft* sheet selected. Using this feature, the user can easily access any of the data fields and modify the data as required.

c. Master-Detail Relationship

On each of the tab sheets, there would usually be a master table; e.g., the *Aircraft* table in Figure 4, and detail tables; e.g., *Acmsn, Acmsnequip, Acewdetprob* and *Aadet*. This arrangement facilitates easy access to related information and minimizes the time required to find the relevant data entries.

d. Views of the Data

The user is shown a *virtual* table; i.e., a *view*, in most of the detail tables presented in the data sheets. This means that the data entries presented in the *virtual* table contain data fields that are not present in the underlying database table. For example, the *Acmsnequip* table only comprise the fields *acmsnid, weaponid* and *qty* but the display on the lower left of Figure 4 also shows the field *descr* which contains the *weapon description* information based on the *weaponid*. *Views* of the data is more effective than the raw table as the user is able to relate to the information shown. In addition, *views* help to ensure that underlying tables retain its *"normalized"* structure; i.e., a state where the table only contains information on a single data object or tables related to that object.

4. Edit Map Data

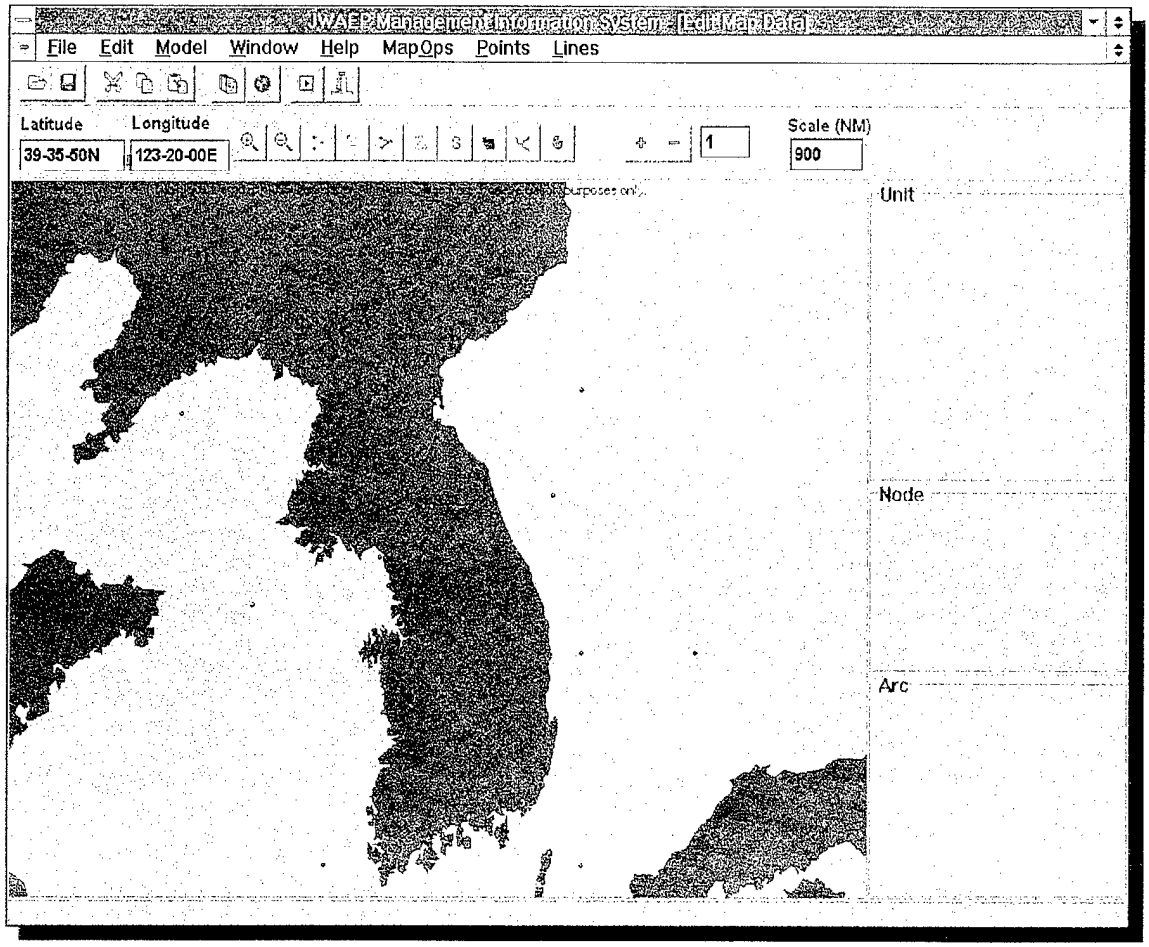

Figure 5 JMIS Main screen showing the Edit Map Data window.

a. Overview

This window is designed to bring together the database elements that have geographical attributes and other database tables that are linked directly to these elements. The prototype currently is able to display data stored in the databases but is unable as yet to enter the data through the interface. Much of the functionality presented in the next few paragraphs are inherited from the inherent capabilities contained in the TWorldmap component[Ref 10]. Without this software component, it would take months to put in place the functionality that has been implemented in the prototype.

b. Resizeable Map Area

The map window, currently showing the Korean Peninsula, has the ability to be resized dynamically during runtime. Resizing functions include *fullscreen*; i.e., displaying the whole world map, *zoom*; i.e., using the mouse to define the zoom area, and *zoombyfactor*; i.e., expanding or contracting the current map area by the specified factor. All these functions allow the user to select the appropriate map resolution.

c. Dynamic Coordinate Positions

The user is able to identify the exact grid coordinate of the locations on the map display. The latitude and longitude coordinates on the map where the screen cursor is located is shown in the edit boxes on the top left of the display.

d. Dynamic Display Settings

The display of objects on the map can be toggled on and off by the user. The user can use the menu functions or toolbar to toggle the settings for *grid lines, nodes, node labels, arcs, state boundaries* and *water bodies.* The map display in Figure 5 shows the *nodes* in the Korean Demonstration Scenario but not *node labels* or *arcs*. The ability to toggle display settings will help to ensure that the interface is not cluttered with non essential items and relevant details are shown.

5. Output Selection

a. Overview

The data management system converts the database information into the plain text files that is required by JWAEP. However, it is likely that only parts of the database will be modified from one run of the scenario to the next. Therefore, it is not efficient to generate the plain text files where no changes have been made to the underlying data and when the "older" plain text file exists. Thus, the idea for the *Output Selection* window is to allow the user to select the desired output options and also to present a set of defaults based on known conditions about the database.

b. Options

There are two main options available for the user; the option to output all the required data files or to output only the selected data files. The first option is quite

straightforward and is the default value when running the program for the very first time; i.e., it assumes that no output data files have been created. The second option is the default once the program has produced an initial set of output files.

c. Selection of Output Files

The process of selecting the files that will be produced by the program is done via status indicators stored in the database. Each of the output files will have two BOOLEAN indicator fields in the *scn (scenario)* table that indicate whether the contents of the file has been modified and whether the file has been previously produced. The modification indicator will be set by other parts of the program that deals with the data input process while the output indicator will be set here. The program will then make use of the information in the indicator fields to set the output options for that specified file. This system will improve performance as the program will only produce a smaller set of output as compared to the case where the program produces all the output files for all runs.

d. Contents of Output Files

The database is expected to grow bigger as more system type information is entered into the system with the increased usage of the model. This means that there is a need to limit the output of the program to relevant system types used for the specific scenario. Otherwise, the output will become voluminous and impact the performance of JWAEP as the model has to do more input and use up system memory to store information not relevant to the current scenario. The process of limiting the output to relevant entries will depend on the creation of virtual datasets. A virtual dataset for *units* in the current scenario can be created using the following SQL commands:

```
CREATE VIEW CURSCNUNIT(UNITTYPE) AS
SELECT DISTINCT UNITID FROM SCNUNIT WHERE
SCN = <CURRENT SCN>;
```

This virtual dataset will be used during the output stage as follows:

```
SELECT * FROM UNITTYPE WHERE TYPEID IN CURSCNUNIT;
```

Using this approach, the program will only output the relevant unit types.

6. JWAEP Execution

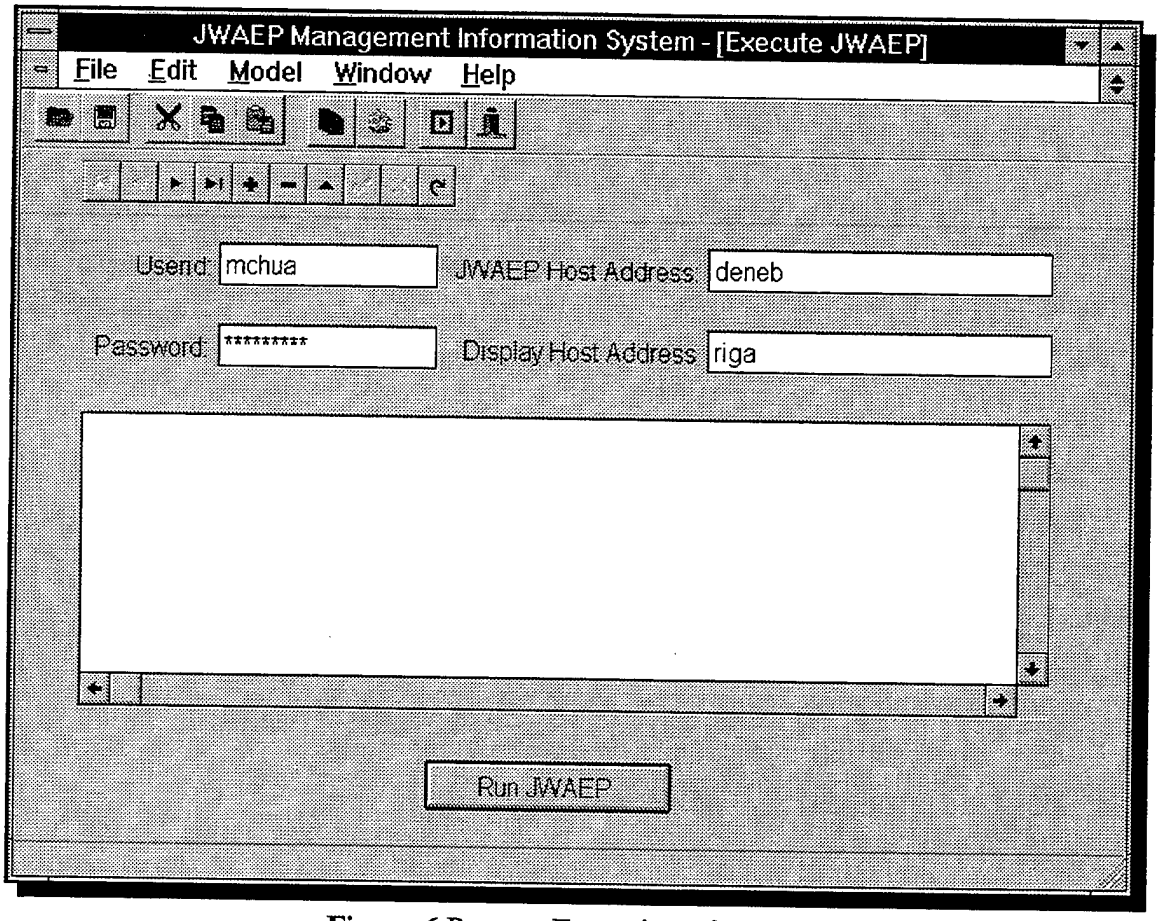

Figure 6 Remote Execution of JWAEP

a. Overview

JMIS controls JWAEP execution via the execute window shown in Figure 6. It essentially allows the user to specify the Unix host where the model reside, the user and password that can access the host and also specify the display server for JWAEP run time graphic. Potentially the user can activate JWAEP from any Unix host that is capable of running the model and have it displayed on the local workstation or remotely on another machine.

b. Implementation

JMIS uses the *rexec* protocol, which essentially allows the Unix workstation to execute remote requests from the network, to trigger JWAEP. A freeware Winsock component is used to implement this functionality in JMIS. The Winsock component[Ref 11] encapsulates the necessary application programming interface(API) calls required for the implementation of the *rexec* protocol to the underlying Windows Sockets dynamic link library(winsock.dll). Winsock.dll is provided as a standard part of the Microsoft Windows 95 and NT operating system.

c. Rexec Command

The *rexec* command used for activating JWAEP is shown in the following example:

setenv DISPLAY riga:0;

cd jwaep2.1;

/usr/openwin/bin/xterm -e jwaep Data/control60.dat;

The first line sets the environmental variable for the location where the graphics are to be displayed. The second line changes the current directory (which defaults to the Unix user home directory) to the JWAEP directory. The third line activates the X terminal program and passes the name of the JWAEP program and control input file to the X terminal as variables[Ref 16].

d. Unix Host Configuration

For the above commands to work, the Unix host must be configured to allow remote execution of programs. In places where strict security standards are enforced, the *rexecd* program may be disabled by default. The *rexecd* command may be made available by the Unix host administrator by adding the following line into the *inetd* file located in the */etc* directory:

exec stream tcp nowait root /usr/sbin/in.rexecd in.rexecd

To prevent unauthorized activation of the *rexec* command, the following line can be added to the *host.allow* file located in the same directory:

in.rexecd: LOCAL, .or.nps.navy.mil 131.120.142.151

This will ensure that only authorized systems are allowed to activate the *rexec* command.

7. BDE Configuration

The interface between the program and the SQL server is defined in the alias JWDATA in the BDE configuration. For the prototype development, the alias is translated to a single Interbase database file, JWDATA, located in the Interbase Server data sub-directory. The path can be easily modified using the BDE configuration utility to point to any other SQL servers or files that has a copy of the database. These changes to the path

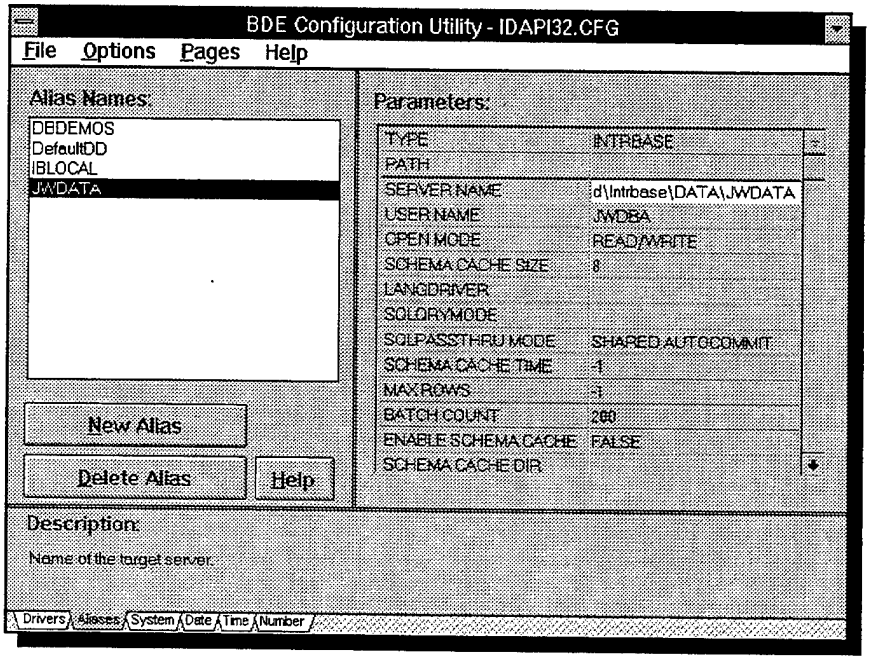

Figure 7 JWDATA alias in the BDE

definition will not affect the operation of JMIS; i.e., the program need not be recompiled.

8. SMB Server

The Samba SMB server is essentially a network file server similar to a NetWare or NT server. Therefore it also requires a certain amount of configuring and administration before

the file sharing services can be made available. The full details[Ref 9] for installing Samba is beyond the scope of this thesis. However the broad steps involved will be presented.

a. Installation

The program can be installed from scratch using the source or from the appropriate binary files available for certain Unix platforms. In this case, the binary files for the SUN OS version 2.1 were used as the source did not compile correctly due to errors in the school's SUN source directories. The binary and configuration files were placed in the default directory locations; i.e., */usr/local/samba*.

b. Configuration

A very simple configuration is used to enable the access to the Unix file system. Essentially the server was configured to allow authorized users to *mount* their home directory. The following is an extract from the *smb.conf*, the Samba configuration file:

[homes]

workgroup = WORKGROUP

browseable = yes

read only = no

create mode = 0750

These parameters essentially tell the Samba server to join the workgroup having the name WORKGROUP, to allow browsing of the shared directories, to allow write access to the shared directory, and to set the default Unix file permissions using the mask 0750. The Unix system administrator can use this file to create other shared directories and set the options according to their requirements.

c. Mapping the Unix Drive to the PC Client

Once the Samba server has been properly installed and configured, the user can map the shared directories to a network drive on the PC. The user can make use of a number of programs, e.g., Windows 95 Explorer, File Manager, *net use* command, etc., to map the network drive. The following is an example of drive mapping using the *net use* command:

net use m: \\sun10or\homes <password>

Once this command is successful, the user can use the mapped drive as if the contents are on the local harddisk. The JMIS program can then write the output file directly into the mapped directory without any additional configuration.

9. X Windows Server

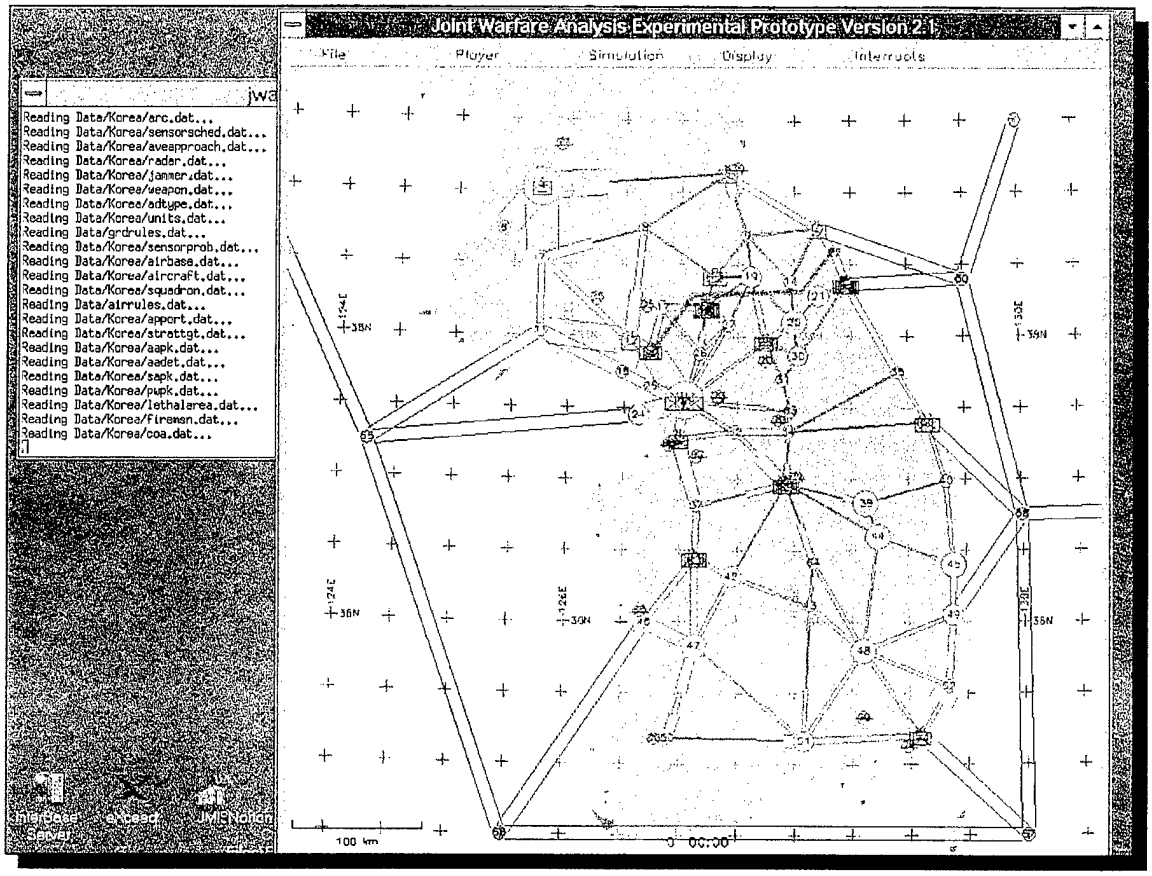

Figure 8 JWAEP Graphic Output Displayed on PC Client

The Exceed X Windows Server is used to facilitate the display of JWAEP run time graphics. The installation of Exceed is a relatively straightforward process and the default values used is sufficient to have a working system. Figure 8 shows the JWAEP model running off the X server in the foreground and the Xterm window, partially covered, running in the background. The JMIS, Exceed server and Interbase server programs are iconized and shown on the bottom left of the screen.

IV. MAINTAINING COMPATIBILITY WITH JWAEP

This chapter discusses the issues of keeping the JMIS program in sync with future developments in JWAEP. As JWAEP is a development model, further enhancements and changes will take place in its development cycle. In fact, the development of version 2.2 is near completion even as JMIS is being developed to meet the database needs for version 2.1. As such, there is a need to identify the changes in JMIS required to keep the program functioning when the user upgrades the JWAEP model.

A. CONSIDERATIONS

The following are some of the issues that have to be looked into:

1. Ability to Use Existing Scenario Data

It is essential that changes to JMIS are backward compatible with existing scenario data files. This means that studies performed using JWAEP version 2.1 scenario files can still be used even if JMIS now supports version 2.2 or higher. This capability will be useful when JWAEP is in use for some time and an archive of project scenarios exists. It may be possible that an old scenario has to be repeated using slightly different variables and it would be annoying if that cannot be supported using the latest version of JMIS except with a major overhaul of the scenario files to incorporate the latest data requirements. Even if the user can convert the files to the new format, the new results would no longer compare to earlier results. Other than being able to use existing data, upgraded versions of JMIS must provide ways for users to convert their existing scenario files into formats that are compatible with newer versions of JWAEP. The conversion process should ideally be automated, though it would take considerable programming effort to do so. At the minimum, there should be ways for the user to manually convert the old scenario data to the new format. This will allow users to take advantage of new model features without losing the investment in the development of the scenario files using the older program.

2. Ability to Continue Using Existing JMIS with New JWAEP Versions

JMIS should be developed such that it is also compatible with future versions of JWAEP without any modifications of the existing program. This way users will be able to continue using JMIS even if the interface or data model have not been updated. However such an approach is only viable as an interim solution as the additional data files will not have been incorporated into the data model and interface, and problems associated with these plain text input data files will remain the responsibility of the user.

3. Concurrent Development

Ultimately the development of JWAEP and JMIS should be done concurrently. The design of the data model is an integral part of the model structure and should be considered when new algorithms and modules are being developed. If that is the case, newer versions of JMIS will not lag behind JWAEP and can be introduced at the same time.

B. IMPLEMENTATION

The following paragraphs look into the areas that have to be modified as we upgrade JMIS to keep it in sync with JWAEP.

1. Tracking Changes

Before we can even discuss what the modifications to the program and data model will be like, we must first have the ability to track these changes and modifications. This piece of information can be contained in the version number field for the scenario data table. Thus all scenarios will bear the version number of the JWAEP model with which they can be run with.

2. Changes to the Data Model

Changes to the data model will be in the form of addition of new tables or modification of existing tables. The addition of new tables, e.g., tables for the implementation of the attack helicopter module, is relatively straightforward and can make use of standard SQL commands similar to that in Appendix 1. The modification for existing tables will use the following SQL commands:

ALTER TABLE <TABLE NAME> ADD <FIELD NAME>

<FIELD TYPE> (OPTIONS);

One point to note in the modification of existing tables is that the changes should not involve removing pre-existing fields. This will help ensure that the data model remains compatible with previous versions of JWAEP.

3. Changes to the User Interface

The user interface will have to be updated to reflect the changes to the data model. The addition of new tables can be easily incorporated into the interface via the addition of new *pages* in the *edit data* window. The output selection window will have to add in options for the these new data tables. Modifications to existing tables mean that the existing interface will have to be similarly modified. This should be an easy process of adding labels and edit boxes for these new entries.

4. Selective Display of Changes

Since some tables and its associated interfaces are used only used by newer versions of the model, JMIS should have the capability to only display the appropriate interface based on the version of the scenario being edited. This means that inappropriate *data pages* will be *greyed out*; i.e., not selectable, whenever they are not part of the required data model. Similarly, fields that are not appropriate can be made *invisible*, by toggling their display property at run time, so that the user is not confused by the additional data requirement.

5. Selective Output

Similar to the user interface, the appropriate output routines will have to selected based on the version number of the scenario. This can be easily incorporated into the output function as shown in the code segment below:

```
if( current_scenario >= 2.2) then
  DoVersion2.2OutputRoutines;
```

The logic for conditional execution of the data output can be more sophisticated than that shown above. If it is more appropriate, the output selection can be determined by using the *case* statements instead.

6. Maintaining Existing Versions of JWAEP

While we can ensure that JMIS is compatible with older version of JWAEP, it would be a pointless exercise if older versions of the model are not available. The location for the JWAEP executable program will be stored in the database and used by the remote execute command. Furthermore, additional information, e.g., new algorithms, data tables, etc., about each version of the program can be maintained within this table, and made available to the user.

7. Using JMIS with New Versions of JWAEP

It would be relatively easy to continue using existing versions of JMIS with updated versions of JWAEP if the updates do not modify original data files; i.e., modifications of the model result in the addition of new data files. If that is the case, JMIS need only read in the names and storage location of the new data files that are required and pass it along with the names and storage location of the other data files in the *control.dat* file to the JWAEP model. Essentially the output routine in the program will look for a file called *include.dat* and append the information contained in the file into *control.dat*. This is a simple but effective way to allow JMIS to be used with newer versions of JWAEP. The drawback is that users will not have the benefit of using JMIS to edit the additional files and the database to provide error checking services. However, this would still be preferable to having to use the plain text editor to modify data files that are already managed by the JMIS program.

V. CONCLUSION

A. SUMMARY

This thesis highlights the problems associated with the current design of the JWAEP data management system and proposes a different approach to overcome them. The viability of the new data management system is demonstrated in the JMIS prototype through the partial implementation of the various proposed program components. Due to the size of the model and the database, it will take more time than available to implement the complete program functionality. It will probably take another six months of development effort to get the JMIS program to a useable level.

B. POSSIBILITIES FOR FURTHER ENHANCEMENTS

The potential for further development in the data management design and prototype exists and will be briefly examined in the following paragraphs.

1. JMIS User Manual

A user manual should be created to help users learn about the JMIS program. This manual will provide the basic information about program operation and describe the program functionality.

2. On-Line Help Files

Context sensitive on-line assistance will be a major benefit to users. One possible area where context sensitive help would be useful is to provide detailed information about database fields. Thus the user can find out the what are the meaningful values that should go into the database fields and perhaps even brief description of the algorithms that use that data.

3. Management of JWAEP Output Files

The focus of the thesis has been the input files that is used by JWAEP. It would be beneficial if the same progress can be applied to the output files. Once the output data are also captured in the JWAEP database, there will be many possibilities for displaying the data to the users.

4. Enhancement of Program Performance

The performance of the JMIS prototype has not been factored in the development process. Thus there is scope for fine tuning the program such that it uses less memory resources and performs adequately for lesser equipped computer systems. For example, the program takes some time to start up as it opens all its data tables during program initialization. This approach may be replaced by selective opening of specified data tables when the user selects the data page. This approach requires more programming effort but will benefit performance and reduce memory requirements.

APPENDIX A.

This appendix contains the SQL statements used to create the initial data model for JMIS. The SQL statements are capitalized while the comments are contained in within these "/* */" symbols.

```
/* New data file */
CREATE TABLE JUSER( USERID VARCHAR(6) NOT NULL PRIMARY KEY,
NAME VARCHAR(20) NOT NULL,
STATUS VARCHAR(5) DEFAULT USER);

CREATE TABLE SCN( SCNID VARCHAR(6) NOT NULL PRIMARY KEY,
DESCR VARCHAR(40) NOT NULL,
JUSER VARCHAR(6) NOT NULL,
MNLONG VARCHAR(10) NOT NULL,
MXLONG VARCHAR(10) NOT NULL,
MNLAT VARCHAR(9) NOT NULL,
MXLAT VARCHAR(9) NOT NULL,
FIRSTCREATED DATE DEFAULT 'now',
LASTMODIFIED DATE DEFAULT 'now',
LASTOPENED DATE DEFAULT 'now',
MAXDIFF FLOAT DEFAULT 0.6,  /* From end of .dat */
MAXDEV FLOAT DEFAULT 2.0,
MODFACTOR FLOAT DEFAULT 0.1,
SUNRISE FLOAT NOT NULL CHECK( SUNRISE <= 1.0 AND SUNRISE >= 0), /* From airrules.dat */
ADTHREATMULT FLOAT DEFAULT 1.0,
AIRDISTMULT FLOAT DEFAULT 1.0,
ICONMOD CHAR(1) DEFAULT 1,
ICONPRN CHAR(1) DEFAULT 0,
ICONPATH VARCHAR(20) DEFAULT "icon.dat",
GRAPHICSMOD CHAR(1) DEFAULT 1,
GRAPHICSPRN CHAR(1) DEFAULT 0,
GRAPHICSPATH VARCHAR(20) DEFAULT "graphics.dat",
CLASSMOD CHAR(1) DEFAULT 1
CLASSPRN CHAR(1) DEFAULT 0,
```

CLASSPATH VARCHAR(20) DEFAULT "class.dat",
ARCMOD CHAR(1) DEFAULT 1,
ARCPRN CHAR(1) DEFAULT 0,
ARCPATH VARCHAR(20) DEFAULT "arc.dat",
NODEMOD CHAR(1) DEFAULT 1,
NODEPRN CHAR(1) DEFAULT 0,
NODEPATH VARCHAR(20) DEFAULT "node.dat",
AVEAPPRMOD CHAR(1) DEFAULT 1,
AVEAPPRPRN CHAR(1) DEFAULT 0,
AVEAPPRPATH VARCHAR(20) DEFAULT 'aveapproach.dat",
SIDEMOD CHAR(1) DEFAULT 1,
SIDEPRN CHAR(1) DEFAULT 0,
SIDEPATH VARCHAR(20) DEFAULT "side.dat",
COAMOD CHAR(1) DEFAULT 1,
COAPRN CHAR(1) DEFAULT 0,
COAPATH VARCHAR(20) DEFAULT "coa.dat",
ALTORDMOD CHAR(1) DEFAULT 1,
ALTORDPRN CHAR(1) DEFAULT 0,
ALTORDPATH VARCHAR(20) DEFAULT "alternord.dat",
EQUIPMOD CHAR(1) DEFAULT 1,
EQUIPPRN CHAR(1) DEFAULT 0,
EQUIPPATH VARCHAR(20) DEFAULT "equipment.dat",
UNITSMOD CHAR(1) DEFAULT 1,
UNITSPRN CHAR(1) DEFAULT 0,
UNITSPATH VARCHAR(20) DEFAULT "units.dat",
GRDRULEMOD CHAR(1) DEFAULT 1,
GRDRULEPRN CHAR(1) DEFAULT 0,
GRDRULEPATH VARCHAR(20) DEFAULT "grdrules.dat",
SITREPMOD CHAR(1) DEFAULT 1,
SITREPPRN CHAR(1) DEFAULT 0,
SITREPPATH VARCHAR(20) DEFAULT "sitrep.dat",
SENSORMOD CHAR(1) DEFAULT 1,
SENSORPRN CHAR(1) DEFAULT 0,
SENSORPATH VARCHAR(20) DEFAULT "sensor.dat",
SENSORSCHMOD CHAR(1) DEFAULT 1,

```
SENSORSCHPRN CHAR(1) DEFAULT 0,
SENSORSCHPATH VARCHAR(20) DEFAULT "sensorsched.dat",
SENSORDETMOD CHAR(1) DEFAULT 1,
SENSORDETPRN CHAR(1) DEFAULT 0,
SENSORDETPATH VARCHAR(20) DEFAULT "sensorprob.dat",
GRIDMOD CHAR(1) DEFAULT 1,
GRIDPRN CHAR(1) DEFAULT 0,
GRIDPATH VARCHAR(20) DEFAULT "grid.dat",
FOREIGN KEY(JUSER) REFERENCES JUSER(USERID));

/* From equipment.dat - BASIC DATA TYPE */
CREATE TABLE EQUIPGP( EQUIPGPID VARCHAR(6) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
STDDEVO SMALLINT NOT NULL,
STDDEVE SMALLINT NOT NULL);

/* Table: SIDE, BASIC DATA TYPE */
CREATE TABLE SYSSIDE (SIDEID VARCHAR(2) NOT NULL PRIMARY KEY,
SIDEDESC VARCHAR(30) NOT NULL);

/* From icon.dat BASIC DATA TYPE */
CREATE TABLE SIZ ( SIZEID VARCHAR(4) NOT NULL PRIMARY KEY,
DESC VARCHAR(30) NOT NULL);

/* From unit.dat BASIC DATA TYPE */
CREATE TABLE UNIT (UNITID VARCHAR(4) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL);

/* From sensor.dat - BASIC DATA TYPE */
CREATE TABLE SENSOR( SENSORID VARCHAR(5) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
TYPE VARCHAR(9) NOT NULL,
ARCPOSERRFCN VARCHAR(2) NOT NULL,
ARCPOSERRP1A FLOAT,
ARCPOSERRP1B FLOAT,
```

```
ARCPOSERRP2A FLOAT,
ARCPOSERRP2B FLOAT,
ARCPOSERRP3A FLOAT,
ARCPOSERRP3B FLOAT,
STRERRFCN VARCHAR(2) NOT NULL,
STRERRP1A FLOAT,
STRERRP1B FLOAT,
STRERRP2A FLOAT,
STRERRP2B FLOAT,
STRERRP3A FLOAT,
STRERRP3B FLOAT,
SPEEDERRFCN VARCHAR(2) NOT NULL,
SPEEDERRP1A FLOAT,
SPEEDERRP1B FLOAT,
SPEEDERRP2A FLOAT,
SPEEDERRP2B FLOAT,
SPEEDERRP3A FLOAT,
SPEEDERRP3B FLOAT,
PROBHDGBACK FLOAT NOT NULL,
TGTERRFCN VARCHAR(2) NOT NULL,
TGTERRP1A FLOAT,
TGTERRP1B FLOAT,
TGTERRP2A FLOAT,
TGTERRP2B FLOAT,
TGTERRP3A FLOAT,
TGTERRP3B FLOAT,
COMMDELAY FLOAT NOT NULL);

/* From class.dat - BASIC DATA TYPE */
CREATE TABLE TERRAIN( TERRAINID VARCHAR(2) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
CLRIDX SMALLINT NOT NULL,
ACCESS SMALLINT NOT NULL);
```

```
/* From weapon.dat - BASIC DATA TYPE */
CREATE TABLE WPN( WPNID VARCHAR(6) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
FCN VARCHAR(2) NOT NULL,
ASSTYPE VARCHAR(5) NOT NULL,
SSM CHAR(1) NOT NULL,
PREC CHAR(1) NOT NULL,
SPEED FLOAT NOT NULL,
MNRANGE FLOAT NOT NULL,
MXRANGE FLOAT NOT NULL,
MXALT FLOAT NOT NULL,
WEIGHT FLOAT NOT NULL,
RADIUS FLOAT NOT NULL,
FIREPREPTM FLOAT NOT NULL,
SHOTTM FLOAT NOT NULL,
MUNITIONS FLOAT NOT NULL);

/* From radar.dat BASIC DATA TYPE */
CREATE TABLE RADAR( RADARID VARCHAR(4) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
RANGE INTEGER NOT NULL,
ALTITUDE INTEGER NOT NULL,
SWEEP.ANGLE SMALLINT NOT NULL,
FCCAPABILITY CHAR(1) NOT NULL,
LNCHRSPERFC SMALLINT NOT NULL,
ACQTMNETTED FLOAT NOT NULL,
ACQTMUNNETTED FLOAT NOT NULL);

/* From unit.dat BASIC DATA TYPE */
CREATE TABLE FCN( FCNID CHAR(2) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL);

/* From aircraft.dat BASIC DATA TYPE */
CREATE TABLE ACTGTCLASS( ACTGTCLASSID VARCHAR(3) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL);
```

```
/* From aircraft.dat BASIC DATA TYPE */
CREATE TABLE ACMSNTYPE( ACMSNTYPEID VARCHAR(8) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL);

/* From ewdetect.dat BASIC DATA TYPE */
CREATE TABLE EWCLASS( EWCLASSID VARCHAR(5) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL);

/* From coa.dat */
CREATE TABLE ORDTYPE( ORDTYPEID VARCHAR(2) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL);

/* From sensor.dat */
CREATE TABLE NETSENSOR( SENSOR VARCHAR(5) NOT NULL,
REPPERIODFCN VARCHAR(2),
REPPERIODP1A FLOAT,
REPPERIODP1B FLOAT,
REPPERIODP2A FLOAT,
REPPERIODP2B FLOAT,
REPPERIODP3A FLOAT,
REPPERIODP3B FLOAT,
FOREIGN KEY(SENSOR) REFERENCES SENSOR(SENSORID));

/* From sensor.dat */
CREATE TABLE SCHSENSOR( SENSOR VARCHAR(5) NOT NULL,
MSNTMFCN VARCHAR(2),
MSNTMP1A FLOAT,
MSNTMP1B FLOAT,
MSNTMP2A FLOAT,
MSNTMP2B FLOAT,
MSNTMP3A FLOAT,
MSNTMP3B FLOAT,
FOOTPRTW FLOAT,
FOOTPRTL FLOAT,
OWNBYBLUE SMALLINT,
```

```
OWNBYRED SMALLINT,
FOREIGN KEY(SENSOR) REFERENCES SENSOR(SENSORID));

/* From equipment.dat */
CREATE TABLE EQUIP( EQUIPID VARCHAR(6) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
MODE VARCHAR(8) NOT NULL,
MXRANGE FLOAT NOT NULL,
PKTSIZE FLOAT NOT NULL,
MXERR FLOAT NOT NULL,
EQUIPGP VARCHAR(6) NOT NULL,
FOREIGN KEY (EQUIPGP) REFERENCES EQUIPGP(EQUIPGPID));

/* From sensorschedule.dat */
CREATE TABLE SENSORSCHD( SCN VARCHAR(6) NOT NULL,
SENSOR VARCHAR(5) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
STARTTM FLOAT NOT NULL,
ENDTM FLOAT NOT NULL,
LATITUDE VARCHAR(9) NOT NULL,
LONGITUDE VARCHAR(10) NOT NULL,
HEADING FLOAT NOT NULL,
PERIODFCN VARCHAR(2) NOT NULL,
PERIODFCNP1A FLOAT NOT NULL,
PERIODFCNP1B FLOAT NOT NULL,
PERIODFCNP2A FLOAT NOT NULL,
PERIODFCNP2B FLOAT NOT NULL,
PRIMARY KEY(SCN, SENSOR),
FOREIGN KEY(SCN) REFERENCES SCN(SCNID),
FOREIGN KEY(SENSOR) REFERENCES SENSOR(SENSORID));

/* From sensorprob.dat */
CREATE TABLE SENSORDETPROB( SENSOR VARCHAR(5) NOT NULL,
EQUIPGP VARCHAR(6) NOT NULL,
TERRAIN VARCHAR(2) NOT NULL,
```

```
DETPROB FLOAT NOT NULL,
FOREIGN KEY(SENSOR) REFERENCES SENSOR(SENSORID),
FOREIGN KEY(EQUIPGP) REFERENCES EQUIPGP(EQUIPGPID),
FOREIGN KEY(TERRAIN) REFERENCES TERRAIN(TERRAINID));

/* From class.dat */
CREATE TABLE UNOPMOVRATE( UNIT VARCHAR(5) NOT NULL,
TERRAIN VARCHAR(2) NOT NULL,
ADMIN FLOAT NOT NULL,
TACTICAL FLOAT NOT NULL,
MVTTOATK FLOAT NOT NULL,
ATTACK FLOAT NOT NULL,
FOREIGN KEY(UNIT) REFERENCES UNIT(UNITID),
FOREIGN KEY(TERRAIN) REFERENCES TERRAIN(TERRAINID));

/* From class.dat */
CREATE TABLE OPMOVRATE( SIZ VARCHAR(6) NOT NULL,
UNIT VARCHAR(5) NOT NULL,
TERRAIN  VARCHAR(2) NOT NULL,
ADMINW FLOAT NOT NULL,
ADMINL FLOAT NOT NULL,
TACTICALW FLOAT NOT NULL,
TACTICALL FLOAT NOT NULL,
MVTTOATKW FLOAT NOT NULL,
MVTTOATKL FLOAT NOT NULL,
ATTACKW FLOAT NOT NULL,
ATTACKL FLOAT NOT NULL,
DEFENDW FLOAT NOT NULL,
DEFENDL FLOAT NOT NULL,
DELAYW FLOAT NOT NULL,
DELAYL FLOAT NOT NULL,
TAAW FLOAT NOT NULL,
TAAL FLOAT NOT NULL,
GENSPTW FLOAT NOT NULL,
GENSPTL FLOAT NOT NULL,
```

```
FOREIGN KEY(SIZ) REFERENCES SIZ(SIZEID),
FOREIGN KEY(UNIT) REFERENCES UNIT(UNITID),
FOREIGN KEY(TERRAIN) REFERENCES TERRAIN(TERRAINID));

/* from Node.dat */
CREATE TABLE NODE( NODEID VARCHAR(5) NOT NULL PRIMARY KEY,
LATITUDE VARCHAR(9) NOT NULL,
LONGITUDE VARCHAR(10) NOT NULL,
RADIUS FLOAT NOT NULL,
TERRAIN VARCHAR(2) NOT NULL,
RSENSOR VARCHAR(5) NOT NULL,
BSENSOR VARCHAR(5) NOT NULL,
REPORT CHAR(1) NOT NULL,
NODEDESC VARCHAR(30) NOT NULL,
FOREIGN KEY(RSENSOR) REFERENCES SENSOR( SENSORID),
FOREIGN KEY(BSENSOR) REFERENCES SENSOR( SENSORID));

/* from arc.dat */
CREATE TABLE ARC( ARCID VARCHAR(5) NOT NULL PRIMARY KEY,
SNODE VARCHAR(5) NOT NULL,
ENODE VARCHAR(5) NOT NULL,
SIDE CHAR(1) NOT NULL, /* Whether which side is aware of the arc */
TERRAIN VARCHAR(2) NOT NULL,
WIDTH SMALLINT NOT NULL,
BSENSOR VARCHAR(5) NOT NULL,
RSENSOR VARCHAR(5) NOT NULL,
FOREIGN KEY(TERRAIN) REFERENCES TERRAIN( TERRAINID),
FOREIGN KEY(RSENSOR) REFERENCES SENSOR( SENSORID),
FOREIGN KEY(BSENSOR) REFERENCES SENSOR( SENSORID));

/* From aveapproach.dat */
CREATE TABLE SCNAVEAPPR( AVEID VARCHAR(6) NOT NULL,
SCN VARCHAR(6) NOT NULL,
DESCR VARCHAR(30) NOT NULL,
NODE VARCHAR(5) [1:10],
```

```
PRIMARY KEY(AVEID, SCN),
FOREIGN KEY(SCN) REFERENCES SCN(SCNID));

/* From aveapproach.dat modified to use arrays above */
CREATE TABLE AVEAPPRNODE( AVE VARCHAR(6) NOT NULL,
NODE VARCHAR(5) NOT NULL,
FOREIGN KEY( NODE) REFERENCES NODE( NODEID));

/* From equipment.dat */
CREATE TABLE EQUIPDEN( EQUIP VARCHAR(6) NOT NULL,
TERRAIN VARCHAR(2) NOT NULL,
ATTACK FLOAT NOT NULL,
PREPDEFEND FLOAT NOT NULL,
HASTYDEFEND FLOAT NOT NULL,
DELAY FLOAT NOT NULL,
MOVE FLOAT NOT NULL,
GENSUPT FLOAT NOT NULL,
NONE FLOAT NOT NULL,
PRIMARY KEY( EQUIP, TERRAIN),
FOREIGN KEY(EQUIP) REFERENCES EQUIP(EQUIPID),
FOREIGN KEY(TERRAIN) REFERENCES TERRAIN(TERRAINID));

/* New File System Data */
CREATE TABLE SYSDATA(
MXADJCYCLE FLOAT DEFAULT 6.0,   /* from groundrules.dat */
MXFIREMSNDURN FLOAT DEFAULT 1.0,
MNTIMEPREPDEF FLOAT DEFAULT 12.0);

CREATE TABLE SCNSIDES( SCN VARCHAR(6) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
DESCR VARCHAR(30) NOT NULL,
CBTSENSOR VARCHAR(5) NOT NULL, /* From sensor.dat */
UPDCYC FLOAT DEFAULT 2.0, /* From side.dat */
COACYC FLOAT DEFAULT 6.0,
COATHR FLOAT DEFAULT 0.8,
```

```
THRMX  SMALLINT DEFAULT 2,
FRPROB FLOAT NOT NULL,
MNFORCERATIOATK FLOAT NOT NULL, /* From grdrules.dat */
MNFORCERATIODEF FLOAT NOT NULL,
MNFRACAUTHSTRWD FLOAT NOT NULL,
MNFRACAUTHSTRDIE  FLOAT NOT NULL,
CLSENOUGHTOMERGE FLOAT NOT NULL,
FIRSTAIRPLANCYC FLOAT NOT NULL, /* From airrules.dat */
LENGTHAIRPLANCYC SMALLINT NOT NULL,
TMOVERTGTSPACING FLOAT NOT NULL,
INTERCEPTRATIO FLOAT NOT NULL,
DCAORBITTM FLOAT NOT NULL,
FSWPORBITTM FLOAT NOT NULL,
MNDCA SMALLINT NOT NULL,
MXDCA SMALLINT NOT NULL,
MNFSWP SMALLINT NOT NULL,
MXFSWP SMALLINT NOT NULL,
MNCAS SMALLINT NOT NULL,
MXCAS SMALLINT NOT NULL,
MNAI SMALLINT NOT NULL,
MXAI SMALLINT NOT NULL,
MNSTI SMALLINT NOT NULL,
MXSTI SMALLINT NOT NULL,
FSWPCASPRIOR SMALLINT NOT NULL,
FSWPAIPRIOR SMALLINT NOT NULL,
FSWPSTIPRIOR SMALLINT NOT NULL,
ANTICIPATEDENEMYADPK FLOAT NOT NULL,
TYPEQUIVALENT VARCHAR(4) NOT NULL,  /* From the end of unit.dat */
SITREPINTERVAL FLOAT NOT NULL, /* From sitrep.dat */
ARCPOSERRFCN VARCHAR(2) NOT NULL,
ARCPOSERRP1A FLOAT,
ARCPOSERRP1B FLOAT,
ARCPOSERRP2A FLOAT,
ARCPOSERRP2B FLOAT,
ARCPOSERRP3A FLOAT,
```

```
ARCPOSERRP3B FLOAT,
STRERRFCN VARCHAR(2) NOT NULL,
STRERRP1A FLOAT,
STRERRP1B FLOAT,
STRERRP2A FLOAT,
STRERRP2B FLOAT,
STRERRP3A FLOAT,
STRERRP3B FLOAT,
SPEEDERRFCN VARCHAR(2) NOT NULL,
SPEEDERRP1A FLOAT,
SPEEDERRP1B FLOAT,
SPEEDERRP2A FLOAT,
SPEEDERRP2B FLOAT,
SPEEDERRP3A FLOAT,
SPEEDERRP3B FLOAT,
PROBHDGBACK FLOAT NOT NULL,
COMMDELAY FLOAT NOT NULL,
PRIMARY KEY(SCN, SIDE),
FOREIGN KEY(SCN) REFERENCES SCN(SCNID),
FOREIGN KEY(SIDE) REFERENCES SYSSIDE(SIDEID),
FOREIGN KEY(ARCPOSERRFCN) REFERENCES STATFCN(FCNID),
FOREIGN KEY(STRERRFCN) REFERENCES STATFCN(FCNID),
FOREIGN KEY(SPEEDERRFCN) REFERENCES STATFCN(FCNID),
FOREIGN KEY(CBTSENSOR) REFERENCES SENSOR(SENSORID),
FOREIGN KEY(TYPEEQUIVALENT) REFERENCES UNITTYPE(TYPEID));

/* From graphics.dat */
CREATE TABLE GRAPHDISP( SCN VARCHAR(6) NOT NULL,
JUSER VARCHAR(10) NOT NULL,
MAP CHAR(1) DEFAULT 0,
ANIMINT FLOAT DEFAULT 1.0,
ARCLABEL CHAR(1) DEFAULT 0,
ARCSTYLE CHAR(1) DEFAULT 1,
NODELABEL CHAR(1) DEFAULT 1,
NODESTYLE VARCHAR(2) DEFAULT 4,
```

```
NODESIZE SMALLINT DEFAULT 250,
CONNECTORNODE CHAR(1) DEFAULT 0,
UNITLABEL CHAR(1) DEFAULT 0,
BATTLE CHAR(1) DEFAULT 1,
BATTLECOLOR VARCHAR(2) DEFAULT 14,
ROUTECOLOR VARCHAR(2) DEFAULT 15,
ROUTESTYLE VARCHAR(2) DEFAULT 1,
FLIGHT CHAR(1) DEFAULT 1,
ENGAGEMENT CHAR(1) DEFAULT 1,
FIREMISSION CHAR(1) DEFAULT 1,
GRIDSW CHAR(1) DEFAULT 0,
GRIDCLR VARCHAR(2) DEFAULT 1,
GRIDSTY VARCHAR(2) DEFAULT 3,
LATLONGSW CHAR(1) DEFAULT 0,
LATLONGCLR VARCHAR(2) DEFAULT 13,
LATLONGMARKSDEG FLOAT DEFAULT 2.0,
LATLONGLABELSDEG FLOAT DEFAULT 0.5,
BLUEPRIMARYCOLOR VARCHAR(2) DEFAULT 5,
REDPRIMARYCOLOR VARCHAR(2) DEFAULT 6,
OCEANCOLOR VARCHAR(2) DEFAULT 18,
ISLANDCOLOR VARCHAR(2) DEFAULT 0,
SPECIALAREACOLOR VARCHAR(2) DEFAULT 17,
POLITBORDERCOLOR VARCHAR(2) DEFAULT 16,
RIVERCOLOR VARCHAR(2) DEFAULT 18,
SPECIALLINECOLOR VARCHAR(2) DEFAULT 17,
MAXSCREENSCALE FLOAT DEFAULT 2.20,
ZOOMFACTORFORMAXSCALE SMALLINT DEFAULT 12,
ZOOMFACTORFORMINSCALE SMALLINT DEFAULT 1,
MXICONSCALE SMALLINT DEFAULT 12,
MNICONSCALE SMALLINT DEFAULT 2,
FOREIGN KEY(SCN) REFERENCES SCN(SCNID),
FOREIGN KEY(JUSER) REFERENCES JUSER(USERID));
```

```
/* From jammer.dat */
CREATE TABLE JAMMER(JAMMERID VARCHAR(4) NOT NULL PRIMARY KEY,
RADAR VARCHAR(4) NOT NULL,
EFFECT SMALLINT NOT NULL,
FOREIGN KEY( RADAR) REFERENCES RADAR(RADARID));

/* From adtype.dat */
CREATE TABLE SECADTYPE( TYPEID VARCHAR(3) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
EQUIP VARCHAR(6) NOT NULL,
WPN VARCHAR(6) NOT NULL,
RNDSPERLAUNCHER SMALLINT NOT NULL,
CUEDRANGE SMALLINT NOT NULL,
UNCUEDRANGE SMALLINT NOT NULL,
RNDAVAILPROB FLOAT NOT NULL,
REFIRETIME FLOAT NOT NULL,
NIGHTCAPABLE CHAR(1) NOT NULL,
FOREIGN KEY(EQUIP) REFERENCES EQUIP(EQUIPID),
FOREIGN KEY(WPN) REFERENCES WPN(WPNID));

/* From sensor.dat */
CREATE TABLE PRIADTYPE( TYPEID VARCHAR(4) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
FCRADAR VARCHAR(4) NOT NULL,
FCQTY SMALLINT NOT NULL,
ACRADAR VARCHAR(4) NOT NULL,
ACQTY SMALLINT NOT NULL,
EQUIP VARCHAR(4) NOT NULL,
EQUIPQTY SMALLINT NOT NULL,
EQUIPQTYSTDDEV SMALLINT NOT NULL,
WPN VARCHAR(4) NOT NULL,
WPNMXRD SMALLINT NOT NULL,
WPNSTDDEV SMALLINT NOT NULL,
TELARFLAG CHAR(1) NOT NULL,
RNDSPERLAUNCHER SMALLINT NOT NULL,
```

```
RNDAVAILPROB FLOAT NOT NULL,
REFIRETIME FLOAT NOT NULL,
RADARREPAIRTM FLOAT NOT NULL,
NIGHTCAPABLE CHAR(1) NOT NULL,
MXPCTAVAILRNDS FLOAT NOT NULL,
FOREIGN KEY( FCRADAR) REFERENCES RADAR(RADARID),
FOREIGN KEY( ACRADAR) REFERENCES RADAR(RADARID),
FOREIGN KEY( EQUIP) REFERENCES EQUIP(EQUIPID),
FOREIGN KEY( WPN) REFERENCES WPN(WPNID));

/* From sensor.dat */
CREATE TABLE ADPRIANDSEC( PRIMARYAD VARCHAR(4) NOT NULL,
SECONDARY VARCHAR(3) NOT NULL,
QTY SMALLINT NOT NULL,
QTYSTDDEV SMALLINT NOT NULL,
MXRND SMALLINT NOT NULL,
MXRNDSTDDEV SMALLINT NOT NULL,
FOREIGN KEY(PRIMARYAD) REFERENCES PRIADTYPE( TYPEID),
FOREIGN KEY(SECONDARY) REFERENCES SECADTYPE( TYPEID));

/* From unit.dat */
CREATE TABLE UNITGP (GPID VARCHAR(5) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
SIZ VARCHAR(4) NOT NULL,
UNITCAT VARCHAR(4) NOT NULL,
FOREIGN KEY(SIDE) REFERENCES SYSSIDE(SIDEID),
FOREIGN KEY(SIZ) REFERENCES SIZ(SIZEID),
FOREIGN KEY(UNITCAT) REFERENCES UNIT(UNITID));

/* From unit.dat */
CREATE TABLE UNITTYPE (TYPEID VARCHAR(6) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
FCN VARCHAR(1) NOT NULL,
MXSUPRANGE SMALLINT NOT NULL,
```

```
GP VARCHAR(5) NOT NULL,
AD VARCHAR(4) NOT NULL,
RADIUS SMALLINT NOT NULL,
FOREIGN KEY (GP) REFERENCES UNITGP(GPID),
FOREIGN KEY (FCN) REFERENCES FCN(FCNID),
FOREIGN KEY (AD) REFERENCES PRIADTYPE(TYPEID));

/* From unit.dat */
CREATE TABLE UNITHIER( SUPUNIT VARCHAR(6) NOT NULL,
SUBUNIT VARCHAR(5) NOT NULL,
QTY SMALLINT NOT NULL,
FOREIGN KEY (SUPUNIT) REFERENCES UNITTYPE(TYPEID),
FOREIGN KEY (SUBUNIT) REFERENCES UNITTYPE(TYPEID));

/* From unit.dat */
CREATE TABLE UNITEQUIP( UNITTYPE VARCHAR(6) NOT NULL,
EQUIP VARCHAR(6) NOT NULL,
QUANTITY SMALLINT NOT NULL,
STDDEV SMALLINT NOT NULL,
FOREIGN KEY(UNITTYPE) REFERENCES UNITTYPE(TYPEID),
FOREIGN KEY(EQUIP) REFERENCES EQUIP(EQUIPID));

/* From unit.dat */
CREATE TABLE SCNUNIT( SCN VARCHAR(6) NOT NULL,
UNITID VARCHAR(5) NOT NULL,
DESCR VARCHAR(30) NOT NULL,
TYPE VARCHAR(4),
PRIMARY KEY(SCN, UNIT),
FOREIGN KEY(SCN) REFERENCES SCN(SCNID),
FOREIGN KEY(TYPE) REFERENCES UNITTYPE(TYPEID));

/* From airbase.dat */
CREATE TABLE AIRBASE( AIRBASEID VARCHAR(4) NOT NULL PRIMARY KEY,
NODE VARCHAR(5) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
```

```
ICON CHAR(1) NOT NULL,
CREWARMFUEL SMALLINT NOT NULL,
CREWSHORTTERM SMALLINT NOT NULL,
CREWLONGTERM SMALLINT NOT NULL,
FOREIGN KEY(NODE) REFERENCES NODE(NODEID),
FOREIGN KEY(SIDE) REFERENCES SYSSIDE(SIDEID));

/* New table */
CREATE TABLE SCNAIRBASE( SCN VARCHAR(6) NOT NULL,
AIRBASE VARCHAR(4),
PRIMARY KEY(SCN, AIRBASE),
FOREIGN KEY(SCN) REFERENCES SCN(SCNID),
FOREIGN KEY(AIRBASE) REFERENCES AIRBASE(AIRBASEID));

/* From aircraft.dat */
CREATE TABLE AIRCRAFT( AIRCRAFTID VARCHAR(5) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
RANGE SMALLINT NOT NULL,
ALERTLAUNCHDEL SMALLINT NOT NULL,
MSNTAKEOFFDEL SMALLINT NOT NULL,
MNFLTSIZ SMALLINT NOT NULL,
DAYBLK SMALLINT NOT NULL,
NIGHTBLK SMALLINT NOT NULL,
MXAAENG SMALLINT NOT NULL,
ICON SMALLINT NOT NULL,
RCS FLOAT NOT NULL,
RADAR VARCHAR(4) NOT NULL,
TGTCLASS VARCHAR(3) NOT NULL,
FIXEDROTARY VARCHAR(6) NOT NULL,
PROBSHORTREP FLOAT NOT NULL,
PROBLONGREP FLOAT NOT NULL,
TMREARM SMALLINT NOT NULL,
TMSHORTREP FLOAT NOT NULL,
TMLONGREP FLOAT NOT NULL,
```

```
ALTLOWDASH SMALLINT NOT NULL,
ALTLOWPEN SMALLINT NOT NULL,
ALTHIGHDASH SMALLINT NOT NULL,
ALTHIGHPEN SMALLINT NOT NULL,
ALTHIGHCRUISE SMALLINT NOT NULL,
ALTORBIT SMALLINT NOT NULL,
ALTMX SMALLINT NOT NULL,
SPTLOWDASH SMALLINT NOT NULL,
SPTLOWPEN SMALLINT NOT NULL,
SPTHIGHDASH SMALLINT NOT NULL,
SPTHIGHPEN SMALLINT NOT NULL,
SPTHIGHCRUISE SMALLINT NOT NULL,
SPTORBIT SMALLINT NOT NULL,
MSNEFFDCA SMALLINT NOT NULL,
MSNEFFFSWP SMALLINT NOT NULL,
MSNEFFCAS SMALLINT NOT NULL,
MSNEFFAI SMALLINT NOT NULL,
MSNEFFSTI SMALLINT NOT NULL,
MSNEFFRESV SMALLINT NOT NULL,
MSNALTDCA CHAR(1) NOT NULL,
MSNALTFSWP CHAR(1) NOT NULL,
MSNALTCAS CHAR(1) NOT NULL,
MSNALTAI CHAR(1) NOT NULL,
MSNALTSTI CHAR(1) NOT NULL,
MSNALTRESV CHAR(1) NOT NULL,
PROBTGTACQCAS FLOAT NOT NULL,
PROBTGTACQAI FLOAT NOT NULL,
PROBTGTACQSTI FLOAT NOT NULL);

/* From aircraft.dat */
CREATE TABLE ACEWDETPROB( AC VARCHAR(5) NOT NULL,
EW VARCHAR(5) NOT NULL,
PROB FLOAT NOT NULL,
FOREIGN KEY(AC) REFERENCES AIRCRAFT(AIRCRAFTID),
FOREIGN KEY(EW) REFERENCES EWCLASS(EWCLASSID));
```

```
/* From aircraft.dat */
CREATE TABLE ACMSN( ACMSNID VARCHAR(4) NOT NULL PRIMARY KEY,
AC VARCHAR(5) NOT NULL,
MSNTYPE VARCHAR(8) NOT NULL,
LAUNCHAAENG SMALLINT NOT NULL,
DEGREECC FLOAT NOT NULL,
PROBJAMON FLOAT NOT NULL,
JAMMER VARCHAR(4) NOT NULL,
FOREIGN KEY(AC) REFERENCES AIRCRAFT(AIRCRAFTID),
FOREIGN KEY(MSNTYPE) REFERENCES ACMSNTYPE(ACMSNTYPEID),
FOREIGN KEY(JAMMER) REFERENCES JAMMER(JAMMERID));

/* From aircraft.dat */
CREATE TABLE ACMSNEQUIP( ACMSNID VARCHAR(4) NOT NULL,
WEAPON VARCHAR(4) NOT NULL,
QTY SMALLINT NOT NULL,
FOREIGN KEY(ACMSNID) REFERENCES ACMSN(ACMSNID),
FOREIGN KEY(WEAPON) REFERENCES WPN(WPNID));

/* From squadron.dat */
CREATE TABLE SQUADRON( SQUADRONID VARCHAR(5) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
AC VARCHAR(5) NOT NULL,
QTY SMALLINT NOT NULL,
SORTIERATE FLOAT NOT NULL,
MSNEFFDCA SMALLINT NOT NULL,
MSNEFFFSWP SMALLINT NOT NULL,
MSNEFFCAS SMALLINT NOT NULL,
MSNEFFAI SMALLINT NOT NULL,
MSNEFFSTI SMALLINT NOT NULL,
MSNEFFRESV SMALLINT NOT NULL,
FOREIGN KEY(AC) REFERENCES AIRCRAFT(AIRCRAFTID));

/* From squadron.dat */
CREATE TABLE SCNSQUADRON( SCN VARCHAR(6) NOT NULL,
```

```
SQUADRON VARCHAR(5) NOT NULL,
AIRBASE VARCHAR(4) NOT NULL,
ARRTM FLOAT NOT NULL,
PLANRANK SMALLINT NOT NULL,
FOREIGN KEY(SCN,AIRBASE) REFERENCES SCNAIRBASE(SCN,AIRBASE),
FOREIGN KEY(SQUADRON) REFERENCES SQUADRON(SQUADRONID)):

/* From airapportionment.dat */
CREATE TABLE AIRAPPORT( SCN VARCHAR(6) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
TIME FLOAT NOT NULL,
DCA SMALLINT NOT NULL,
FSWP SMALLINT NOT NULL,
CAS SMALLINT NOT NULL,
AI SMALLINT NOT NULL,
STI SMALLINT NOT NULL,
RESERVE SMALLINT NOT NULL,
FOREIGN KEY(SCN,SIDE) REFERENCES SCNSIDES(SCN,SIDE));

/* From strattgt.dat */
CREATE TABLE STRATTGT( SCN VARCHAR(6) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
NODE VARCHAR(5) NOT NULL,
FREQFCN VARCHAR(1) NOT NULL,
FREQP1A FLOAT NOT NULL,
FREQP1B FLOAT NOT NULL,
FREQP2A FLOAT NOT NULL,
FREQP2B FLOAT NOT NULL,
FREQP3A FLOAT NOT NULL,
FREQP3B FLOAT NOT NULL,
PRIORFCN VARCHAR(1) NOT NULL,
PRIOR1A FLOAT NOT NULL,
PRIOR1B FLOAT NOT NULL,
PRIOR2A FLOAT NOT NULL,
PRIOR2B FLOAT NOT NULL,
```

```
PRIOR3A FLOAT NOT NULL,
PRIOR3B FLOAT NOT NULL,
SORTIEDDFCN VARCHAR(1) NOT NULL,
SORTIE1A FLOAT NOT NULL,
SORTIE1B FLOAT NOT NULL,
SORTIE2A FLOAT NOT NULL,
SORTIE2B FLOAT NOT NULL,
SORTIE3A FLOAT NOT NULL,
SORTIE3B FLOAT NOT NULL,
FOREIGN KEY(NODE) REFERENCES NODE(NODEID),
FOREIGN KEY(SCN,SIDE) REFERENCES SCNSIDES(SCN,SIDE));

/* From sapk.dat */
CREATE TABLE SECADPK( AD VARCHAR(3) NOT NULL,
TGTCLASS VARCHAR(3) NOT NULL,
PK FLOAT NOT NULL,
FOREIGN KEY(AD) REFERENCES SECADTYPE(TYPEID),
FOREIGN KEY(TGTCLASS) REFERENCES ACTGTCLASS(ACTGTCLASSID));

/* From sapk.dat */
CREATE TABLE PRICADPK( AD VARCHAR(3) NOT NULL,
TGTCLASS VARCHAR(4) NOT NULL,
PK FLOAT NOT NULL,
FOREIGN KEY(AD) REFERENCES PRIADTYPE(TYPEID),
FOREIGN KEY(TGTCLASS) REFERENCES ACTGTCLASS(ACTGTCLASSID));

/* From aadet.dat */
CREATE TABLE ACDETECT( AC1 VARCHAR(5) NOT NULL,
AC2 VARCHAR(5) NOT NULL,
PRODDETECT FLOAT NOT NULL,
FOREIGN KEY(AC1) REFERENCES AIRCRAFT(AIRCRAFTID),
FOREIGN KEY(AC2) REFERENCES AIRCRAFT(AIRCRAFTID));
```

```
/* From aapk.dat */
CREATE TABLE ACWPNCOMBO( COMBOID VARCHAR(3) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
WPN VARCHAR(4) NOT NULL,
AC VARCHAR(5) NOT NULL,
FOREIGN KEY(WPN) REFERENCES WPN(WPNID),
FOREIGN KEY(AC) REFERENCES AIRCRAFT(AIRCRAFTID));

/* From aapk.dat */
CREATE TABLE COMBOPK( COMBO VARCHAR(3) NOT NULL,
TGT VARCHAR(3) NOT NULL,
PK FLOAT NOT NULL,
FOREIGN KEY(COMBO) REFERENCES ACWPNCOMBO(COMBOID),
FOREIGN KEY(TGT) REFERENCES ACTGTCLASS(ACTGTCLASSID));

/* From aapk.dat */
CREATE TABLE COMBOADV( COMBO VARCHAR(3) NOT NULL,
COMBOTHREAT VARCHAR(3) NOT NULL,
ADV SMALLINT NOT NULL,
FOREIGN KEY(COMBO) REFERENCES ACWPNCOMBO(COMBOID),
FOREIGN KEY(COMBOTHREAT) REFERENCES ACWPNCOMBO(COMBOID));

/* From pwpk.dat Pt Wpn */
CREATE TABLE PTWPNPKEQUIP( WPN VARCHAR(4) NOT NULL,
EQUIP VARCHAR(4) NOT NULL,
PK FLOAT NOT NULL,
FOREIGN KEY(WPN) REFERENCES WPN(WPNID),
FOREIGN KEY(EQUIP) REFERENCES EQUIP(EQUIPID));

/* From pwpk.dat Pt Wpn */
CREATE TABLE PTWPNPKRADAR( WPN VARCHAR(4) NOT NULL,
RADAR VARCHAR(4) NOT NULL,
PK FLOAT NOT NULL,
FOREIGN KEY(WPN) REFERENCES WPN(WPNID),
FOREIGN KEY(RADAR) REFERENCES RADAR(RADARID));
```

```
/* From lethalarea.dat */
CREATE TABLE LETHALAREAEQUIP( WPN VARCHAR(4) NOT NULL,
EQUIP VARCHAR(4) NOT NULL,
AREA SMALLINT NOT NULL,
FOREIGN KEY(WPN) REFERENCES WPN(WPNID),
FOREIGN KEY(EQUIP) REFERENCES EQUIP(EQUIPID));

/* From lethalarea.dat */
CREATE TABLE LETHALAREARADAR( WPN VARCHAR(4) NOT NULL,
RADAR VARCHAR(4) NOT NULL,
AREA SMALLINT NOT NULL,
FOREIGN KEY(WPN) REFERENCES WPN(WPNID),
FOREIGN KEY(RADAR) REFERENCES RADAR(RADARID));

/* From firemsn.dat */
CREATE TABLE FIREMSN( WPN VARCHAR(4) NOT NULL,
UNIT VARCHAR(5) NOT NULL,
ROUNDS SMALLINT NOT NULL,
RADIUS SMALLINT NOT NULL,
FOREIGN KEY(WPN) REFERENCES WPN(WPNID),
FOREIGN KEY(UNIT) REFERENCES UNITTYPE(TYPEID));

/* From coa.dat */
CREATE TABLE COA( SCN VARCHAR(6) NOT NULL,
SIDE VARCHAR(2) NOT NULL,
COAID VARCHAR(6) NOT NULL PRIMARY KEY,
DESCR VARCHAR(30) NOT NULL,
AVEAPPR VARCHAR(4) NOT NULL,
FOREIGN KEY(SCN,SIDE) REFERENCES SCNSIDES(SCN,SIDE),
FOREIGN KEY(COAID,SCN) REFERENCES SCNAVEAPPR(AVEID,SCN));

/* From coa.dat */
CREATE TABLE COAUNIT( COA VARCHAR(6) NOT NULL,
UNIT VARCHAR(5) NOT NULL,
DELTA FLOAT NOT NULL,
```

```
ORDSEQ CHAR(1) NOT NULL,
ORDTYPE VARCHAR(2) NOT NULL,
PARAM1 SMALLINT NOT NULL,
PARAM2 VARCHAR(6) NOT NULL,
FOREIGN KEY(COA) REFERENCES COA(COAID),
FOREIGN KEY(UNIT) REFERENCES UNITTYPE(UNITTYPEID),
FOREIGN KEY(ORDTYPE) REFERENCES ORDTYPE(ORDTYPEID));

/* From coa.dat */
CREATE TABLE COAUNITNODE( COA VARCHAR(6) NOT NULL,
UNIT VARCHAR(5) NOT NULL,
ORDSEQ CHAR(1) NOT NULL,
NODE VARCHAR(5) [1:10],
FOREIGN KEY(COA) REFERENCES COA(COAID),
FOREIGN KEY(UNIT) REFERENCES UNITTYPE(TYPEID),
FOREIGN KEY(NODE) REFERENCES NODE(NODEID));

/* From alternord.dat - Do not need the side info as it is
embedded in the COA table */
CREATE TABLE ALTORD( COA VARCHAR(6) NOT NULL,
UNIT VARCHAR(5) NOT NULL,
DELTA FLOAT NOT NULL,
ORDSEQ CHAR(1) NOT NULL,
ORDTYPE VARCHAR(2) NOT NULL,
PARAM1 SMALLINT NOT NULL,
PARAM2 VARCHAR(6) NOT NULL,
FOREIGN KEY(COA) REFERENCES COA(COAID),
FOREIGN KEY(UNIT) REFERENCES UNITTYPE(TYPEID),
FOREIGN KEY(ORDTYPE) REFERENCES ORDTYPE(ORDTYPEID));

/* From alternord.dat */
CREATE TABLE ALTCOAUNITNODE( COA VARCHAR(6) NOT NULL,
UNIT VARCHAR(5) NOT NULL,
ORDSEQ CHAR(1) NOT NULL,
NODESEQ CHAR(1) NOT NULL,
```

```
NODE VARCHAR(5) NOT NULL,
FOREIGN KEY(COA) REFERENCES COA(COAID),
FOREIGN KEY(UNIT) REFERENCES UNITTYPE(TYPEID),
FOREIGN KEY(NODE) REFERENCES NODE(NODEID));
```

LIST OF REFERENCES

1. Robert S. Hume, *Modeling Attack Helicopter Operations in Theater Level Simulations*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1996.

2. Models & Simulations: Army Integrated Catalog (MOSAIC) (May 96), *JANUS*, URL:http://hp01.arc.iquest.com/mosaic/158.html, January 1996.

3. Titan, Inc. Applications Group, *Janus User's Manual Version 3.X*, Leavenworth, Kansas, March 1993.

4. Titan, Inc. Applications Group, *Janus Data Base Manager's Manual Version 3.X*, Leavenworth, Kansas, March 1993.

5. Special Operations Forces Simulation (June95), *JCM - Joint Conflict Model*, URL:http://hp01.arc.iquest.com/sof/JCM.html.

6. Michael Uzelac et al., *The JCM Simulation Manual*, Lawrence Livermore National Laboratory, Livermore, California, October 1995.

7. Michael Uzelac et al., *The JCM Scenario Editor's Manual*, Lawrence Livermore National Laboratory, Livermore, California, October 1995.

8. Rolands and Associates, *JTLS Analyst Guide Version 1.85*, Monterey, California, April 1994.

9. Andrew Tridgell, *Samba SMB Server Version 1.9.16p2*, URL:*http://lake.canberra.edu.au/pub/samba/*, August 1996.

10. Don Bauer, *TWorldMap Component Version 1.2*, URL:*http://www.stanleyassoc.com/worldmap/wmap.htm/*, August 1996.

11. Gary T. Desrosiers., *Sockets Component Version 3*, URL:*mailto:desrosi@pcnet.com*, March 1996.

12. Youngren, Mark A., Lovell, Neal T., *The Joint Warfare Analysis Experimental Prototype (JWAEP) Version 2.0 User Documentation Draft*, Naval Postgraduate School, Monterey, California, March 1996.

13. Unix Manual Pages, *Ftp (1C)*, Sun Release 4.1, California, January 1988.

14. Unix Manual Pages, *NFS (4B)*, Sun Release 4.1, California, November 1987.

15. Richard Sharpe, *Just what is SMB?*, URL:http://samba.anu.edu.au/cifs/docs/what-is-smb.html, July 1996.

16. Unix Manual Pages., *Xterm (1)*, Massachusetts Institute of Technology, Boston, MA, May 1990.

17. Unix Manual Pages, *Rexec (3B)*, Sun Release 4.1, California, November 1987.

18. Unix Manual Pages, *Rexecd (8C)*, Sun Release 4.1, California, December 1987.

BIBLIOGRAPHY

Borland International, *Delphi User's Guide*, Scotts Valley, California 1996.

Borland International, *Delphi Database Application Developer's Guide*, Scotts Valley, California 1996.

Borland International, *Interbase Server*, Scotts Valley, California 1996.

Borland International, *Online Help - Local SQL Help*, Scotts Valley, California 1996.

Kroenke, D.M., *Database Processing: Rundamentals, Design, Implementation*, Macmillan Publishing Company, 1992.

INITIAL DISTRIBUTION LIST

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

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

3. Chairman, Department of Operations Research 1
   Captain Frank C. Petho
   Code OR/Pe
   Naval Postgraduate School
   Monterey, California 93943

4. Asst. Prof. Mark A. Youngren 10
   Code OR/Ym
   Naval Postgraduate School
   Monterey, California 93943

5. Assoc. Prof. Hemant Bhargava 2
   Code SM/Bh
   Naval Postgraduate School
   Monterey, California 93943

6. Prof. Sam Parry 1
   Code OR/Py
   Naval Postgraduate School
   Monterey, California 93943

7. Prof. Patricia Jacobs 1
   Code OR/Jc
   Naval Postgraduate School
   Monterey, California 93943

8. Prof. Donald Gaver 1
   Code OR/Gv
   Naval Postgraduate School
   Monterey, California 93943

9. Deputy Director for Technical Operations, J-8 1
8000 The Joint Staff
Washington, DC 20318-8000

10. Chief, Warfighting Analysis Division, J-8 2
8000 The Joint Staff
Washington, DC 20318-8000

11. Director, US Army TRADOC Analysis Center 2
ATTN: ATRC (Mr. Bauman)
Ft. Leavenworth, KS 66027-5200

12. Director, US Army TRAC Analysis Center - Monterey 1
ATTN: MAJ Leroy A. Jackson
P.O. Box 8692
Monterey, California 93943-0692

13. Director, US Army TRADOC Analysis Center - Monterey 1
ATTN: ATRC-RDM (LTC Wood)
Monterey, CA 93943

14. Air Force Institute of Technology 1
AFIT/ENS
2950 P St.
Wright-Patterson AFB, OH 45433

15. US Army Concepts Analysis Agency 1
ATTN: Mr. Chandler
8120 Woodmont Ave.
Bethesda, MD 20814-2797

16. US Army War College 1
Center for Strategic Leadership
ATTN: COL Wilkes
Bldg 650
Carlisle, PA 17013-5050

17. The Joint Staff - J6AC 1
ATTN: John Garstka
6000 The Joint Staff
Washington, DC 20318-6000

18. Center for Naval Warfare Studies 1
Naval War College
Newport, RI 02841

19. US Army Missile Command 1
ATTN: AMSMI-RD-AC (Dr. Fowler)
Redstone Arsenal, AL 35898-5242

20. US Army Material Systems Analysis Activity 1
ATTN: Mr. Bill Clay
Aberdeen Proving Ground, MD 21005-5001

21. Dr. Dean Hartley 1
Martin Marietta Energy Systems
1099 Commerce Park
Oak Ridge, TN 37830

22. Dr. Cy Staniec 1
OSD / PA&E
1800 Defense, Pentagon
Washington, DC 20301-1800

23. Mr. Clayton Thomas 1
HQ USAF, AFSAA / SAN
AF Studies & Analysis Agency
1570 Air Force, Pentagon
Washington, DC 20330-1570

24. Mr. Dean Free 1
OCNO (N-81), Department of the Navy
2000 Navy, Pentagon
Washington, DC 20350-2000

25. Mr. Gene Visco 1
SAUS-OR
102 Pentagon, Room 1E643
Washington, DC 20310-0102

26. Dr. Alfred Brandstein 1
MAGTF Warfighting Center
Studies & Analysis Branch (WF13)
Marine Corps Combat Developments Command
Quantico, VA 22134-5001

27. MAJ Michael T. L. Chua 2
554C Joo Chiat Road
Singapore 1542
Republic of Singapore