



Calhoun: The NPS Institutional Archive

Theses and Dissertations

Thesis Collection

1996-03

Design and implementation of a prototype database system for the operational activity schedule of the Hellenic Navy

Marinos, Evangelos Pavlos.

Monterey, California. Naval Postgraduate School



Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

**DESIGN AND IMPLEMENTATION OF A
PROTOTYPE DATABASE SYSTEM FOR THE
OPERATIONAL ACTIVITY SCHEDULE OF THE
HELLENIC NAVY**

by

Evangelos Pavlos Marinos

March 1996

Thesis Advisor:

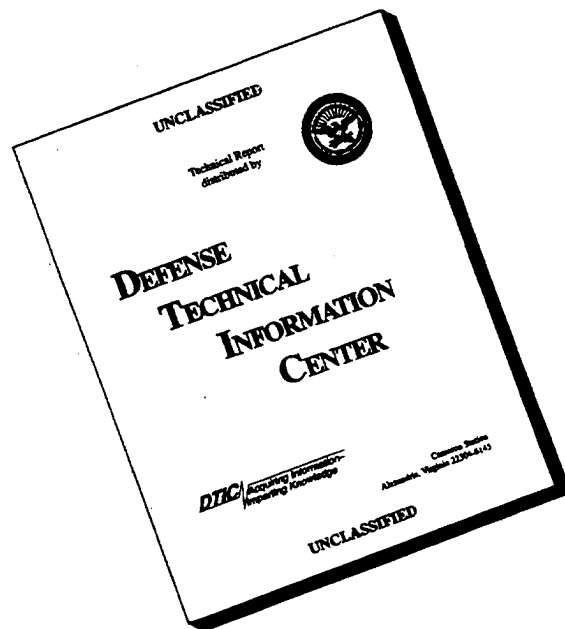
James C. Emery

Approved for public release; distribution is unlimited.

19960708 038

DTIC QUALITY INSPECTED 1

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE March 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE : DESIGN AND IMPLEMENTATION OF A PROTOTYPE DATABASE SYSTEM FOR THE OPERATIONAL ACTIVITY SCHEDULE OF THE HELLENIC NAVY		5. FUNDING NUMBERS	
6. AUTHOR: Evangelos P. Marinos			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) The Hellenic Navy General Staff has a difficult mission which encompasses tactical, operational, and administrative tasks. The most important operational task for the General Staff is to prepare the Operational Activity Schedule for every ship, subcommand, and command in the Hellenic Navy. In order to more effectively prepare this schedule, an automated database is required. This system would contain all operational activity records for the Hellenic Navy units and other pertinent information. Furthermore, the system would produce ad hoc reports, as well as a variety of other reports designed by the user to support ship maintenance schedule. This thesis designs and implements an automated database system that can be used from the Hellenic Navy General Staff. The methodology followed is the standard systems' development life cycle (SDLC). The requirements for the system are obtained, and the database and application are designed and implemented. Paradox 5.0 for Windows is used for the database management system software. Special issues like training, conversion, and maintenance are taken into consideration. The result of this thesis is a functional application named "OADS" (Operational Activity Database System) that will fulfill users' requirements, keeps track of the operational activities of the Hellenic Navy units, and help in performing the desired tasks.			
14. SUBJECT TERMS DESIGN IMPLEMENTATION DATABASE MANAGEMENT SYSTEM SUPPORT OPERATIONAL ACTIVITY SCHEDULE		15. NUMBER OF PAGES 148	
		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
2-89)

Standard Form 298 (Rev.

Prescribed by ANSI Std. Z39-18

Approved for public release; distribution is unlimited.

**DESIGN AND IMPLEMENTATION OF A PROTOTYPE DATABASE SYSTEM
FOR THE OPERATIONAL ACTIVITY SCHEDULE OF THE HELLENIC NAVY**

Evangelos Pavlos Marinos
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

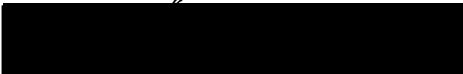
NAVAL POSTGRADUATE SCHOOL

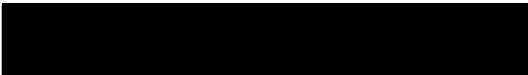
March 1996


Author:


Evangelos P. Marinos

Approved by:


James C. Emery, Thesis Advisor


Magdi N. Kamel, Associate Thesis Advisor


Reuben T. Harris, Chairman
Department of Systems Management

ABSTRACT

The Hellenic Navy General Staff has a difficult mission which encompasses several tactical, operational, and administrative tasks. The most important operational task for the General Staff is to prepare the Operational Activity Schedule for every ship, subcommand, and command in the Hellenic Navy. In order to more effectively prepare this schedule, an automated database system is required. This system would contain all operational activity records for the Hellenic Navy units and other pertinent information. Furthermore, the system would produce ad hoc reports, as well as a variety of other reports designed by the user to support ship maintenance schedule.

This thesis designs and implements an automated database system that can be used from the Hellenic Navy General Staff. The methodology followed is the standard systems' development life cycle (SDLC). The requirements for the system are obtained, and the database and application are designed and implemented. Paradox 5.0 for Windows is used for the database management system software. Special issues like training, security, conversion, and maintenance are taken into consideration.

The result of this thesis is a functional application named "OADS" (Operational Activity Database System) that will fulfill users' requirements, keeps track of the operational activities of the Hellenic Navy units, and help in performing the desired tasks accurately.

TABLE OF CONTENTS

I. INTRODUCTION.....	1
A. OBJECTIVE.....	1
B. BACKGROUND.....	1
C. METHODOLOGY.....	3
D. CHAPTER OUTLINE.....	4
II. DATABASE DEVELOPMENT PROCESS.....	7
A. DATABASE DEFINITIONS.....	7
B. DATABASE DEVELOPMENT METHODOLOGY.....	10
C. REQUIREMENTS ANALYSIS / SPECIFICATIONS.....	11
1. Data Requirements.....	11
2. Entity-Relationship Model.....	12
3. Data Dictionary.....	14
4. Process Requirements.....	15
D. DATABASE DESIGN.....	18
1. Logical Database Design.....	18
2. Application Design.....	20
E. DATABASE IMPLEMENTATION.....	21

III. REQUIREMENTS ANALYSIS FOR OADS.....	23
A. DATA REQUIREMENTS.....	23
1. Entities.....	23
2. Relationships.....	25
B. DATA DICTIONARY.....	26
C. PROCESS REQUIREMENTS.....	27
1. Update Subsystem.....	27
2. Retrieval Subsystem.....	29
D. HARDWARE REQUIREMENTS.....	30
IV. LOGICAL DATABASE AND APPLICATION DESIGN FOR OADS	31
A. LOGICAL DATABASE DESIGN.....	31
B. APPLICATION DESIGN.....	35
V. IMPLEMENTATION FOR OADS.....	37
A. DATA IMPLEMENTATION.....	38
B. APPLICATION IMPLEMENTATION.....	39
VI. OTHER ISSUES.....	41
A. SECURITY.....	41
B. TRAINING.....	42

C. CONVERSION.....	43
D. MAINTENANCE.....	44
E. FUTURE ENHANCEMENTS.....	44
VII. CONCLUSIONS AND LESSONS LEARNED.....	45
APPENDIX A: DATA DICTIONARY.....	47
APPENDIX B: DATA FLOW DIAGRAMS.....	51
APPENDIX C: RELATIONS AND RELATIONAL MODEL.....	67
APPENDIX D: BUSINESS RULES.....	69
APPENDIX E: PROCEDURES FOR INSTALLING AND OPERATING OADS.....	77
APPENDIX F: APPLICATION CODE.....	79
APPENDIX G: APPLICATION MENUS.....	127
LIST OF REFERENCES.....	133

BIBLIOGRAPHY.....135

INITIAL DISTRIBUTION LIST.....137

I. INTRODUCTION

A. OBJECTIVE

This thesis designs and implements a database system for the General Staff of the Hellenic Navy. The purpose of the system is to keep track of all the operational activities of the commands, subcommands, and ships in the Hellenic Navy during a specific period of time. The implementation of the database system would greatly reduce the work hours spent on the preparation of operational programs that are instrumental in accomplishing the principal tasks of commands, subcommands, and ships in the Hellenic Navy. The database design takes into consideration the Hellenic Navy General Staff's functional requirements. The primary function of the database system is to maintain the records of operational activities by command/subcommand/ship and other relevant information. From this database, standard reports are generated and ad hoc queries and reports are created.

B. BACKGROUND

Each year the General Staff of the Hellenic Navy prepares the Operational Activity Schedule for every ship, subcommand, and command in the Hellenic Navy, without taking into consideration the operational activities of these elements in the previous year. Nowadays, the Operational Activity Schedule is being prepared manually

by an office in the General Staff of the Navy. This office keeps data about the operational activities of the ships, subcommands, and commands. Although the system works, it has a number of deficiencies:

- A constant stream of paperwork (in the form of memos, reports, and so on) and telephone calls is required to update the data in the files.
- The system cannot easily provide answers to complex operational questions. For example, answering the question, "Which ship(s) have used over three torpedoes in the drill no. 26?" would probably require some research.
- Senior officers in the General Staff of the Hellenic Navy cannot easily obtain summary information required for decision making.

All the above entail some problems in preparing the Operational Activity Schedule. Many ships, subcommands, and commands that had many activities in the previous year are scheduled to continue having many activities in the following year, leaving other ships, subcommands, and commands relatively idle for two consecutive years. This situation makes the personnel of the busy commands feel that they are unfairly treated by the General Staff of the Navy.

The Operational Activity Database System (OADS) tries to remedy this situation by capturing the operational activities of all the ships, subcommands, and commands, during a one-year period. Reports are generated at the end of the year that include the total hours for each individual ship, subcommand, and command spent in exercises and

individual drills, as well as the hours that a specific ship was at a port. Reports also include information about whether a ship has performed maintenance activities in a port, or if a ship has used any missiles or torpedoes during a drill time.

The General Staff of the Navy will first analyze the OADS's reports and then will prepare the Operational Activity Schedule for the activities of the next year.

C. METHODOLOGY

There are different methodologies for developing application systems. The process that will be followed in this thesis captures the essence of most development methodologies, as they are described in the text of Kroenke [Ref. 1]. The fundamental phases are:

- *Definition phase.* During the definition phase, the tasks are to form the working team, define the problem, establish the scope, and access feasibility issues.
- *Requirements phase.* During the requirements phase, the tasks are to create the user's data model; determine the update, display, and control mechanisms; and determine the functional components of the application. This is accomplished by interviewing the users and by using prototypes to help determine user requirements.

- *Evaluation phase.* During the evaluation phase, the tasks are to select the system's architecture, and reassess feasibility issues.
- *Design phase.* During the design phase, the tasks are to develop the database design and the application design. The database design consists of structuring the relations and establishing the relationships among them. The application design deals with the design of the menus, reports, and forms, as well as to specifying update, display, and control mechanisms.
- *Implementation phase.* During the implementation phase, the tasks are to construct the database, build the application, and install it.

This System's Development Life Cycle was utilized in the development of the Operational Activity Database System (OADS) of this thesis. The organization of this thesis is identical to the organization that was used in the development of a Database System for the Hellenic Navy by Tsongas [Ref. 2], due to the similarity of the scope of the two applications and the common limitations of the infrastructure in the Hellenic Navy units.

D. CHAPTER OUTLINE

In this thesis, Chapters II - VI have the same organization and emphasize the same issues as the thesis of Tsongas [Ref. 2]. Specifically, this thesis is organized as follows:

Chapter II is a general description of the database development process, as it is described in Kroenke [Ref. 1]. It reviews database concepts and describes the database development phases. These phases are detailed in the following chapters as they apply to the OADS application.

Chapter III discusses the requirements analysis for the application system. The operating environment is studied by means of the user's descriptive list of requirements for the system's functionality, data manipulation, and production of specific information. The requirements and accompanying entity-relationship data flow diagrams are provided. The chapter concludes with a description of the requirements specifications as they pertain to data, hardware, and software issues.

Chapter IV describes the design process followed in developing the Operational Activity Database System (OADS). The data and process models developed in the previous chapters are transformed into a relational and application design, respectively. The last section provides commentary about the data dictionary and its benefits to the database system design.

Chapter V is a discussion of the final phases involved in developing the database system. These phases are the implementation portion of the data and process design, and include programming and planning for the system's implementation.

Chapter VI deals with other important issues in developing the system, such as database security, personnel training, system conversion, maintenance, and future upgrades.

Chapter VII is the concluding chapter. It provides a short summary of the thesis and addresses future enhancements to the system developed. Also included are lessons learned in developing the system.

Appendices A through G supplement the previously described text. The appendices are: Data Dictionary, Data Flow Diagrams, Relations and Relational Schema, Business Rules, Application Code, Application Menus, and Procedures for installing and operating OADS.

II. DATABASE DEVELOPMENT PROCESS

Basic database definitions as well as the database development methodology are presented in this chapter. Each step of the system's development methodology is described in some detail. The discussion of this chapter is largely based on the texts of Kroenke [Ref. 1] and Whitten [Ref. 3], and follows the same organization of the thesis of Tsongas [Ref. 2].

A. DATABASE DEFINITIONS

Database is a set of related records. The term *database* has been used to refer to everything from a collection of index cards to the volumes of data that a government collects about its citizens. In the following, we shall use this term with a specific meaning, as it is indicated in Kroenke [Ref. 1]: *A database is a self-describing collection of integrated records.*

1. A Database Is Self-Describing

As Kroenke points out in [Ref. 1], a database contains a description of its own structure, in addition to the user's source data. This description is called the data dictionary (or data directory, or metadata), and makes program/data independence possible. By examining the database itself, it is easy to determine its structure and its components; no external documentation of file and record formats is needed. In addition to that, if we change the structure of the data in the database (such as inserting new data

items to an existing record), we enter only that change in the data dictionary. Few, if any, programs will need to be changed. In most cases, only those programs that process the altered data items must be changed.

2. A Database Is a Collection of Integrated Records

The standard hierarchy of data is as follows: Bits are aggregated into bytes or characters; characters are aggregated into fields; fields are aggregated into records; and records are aggregated into files. Following the pattern of that statement, files are aggregated into databases. [Ref. 1: p.14]

A database includes files of user and other data. As mentioned earlier, a database contains a description of itself in the form of metadata. In addition, a database can include indexes that are used to represent relationships among the data and also to improve the performance of database applications. Finally, the database often contains data about the applications that use the database. The structure of a data entry form, or a report, is sometimes part of the database. This last category of data is called application metadata. Thus a database contains the four types of data: files of user data, metadata, indexes, and application metadata (data about the applications that use the database).

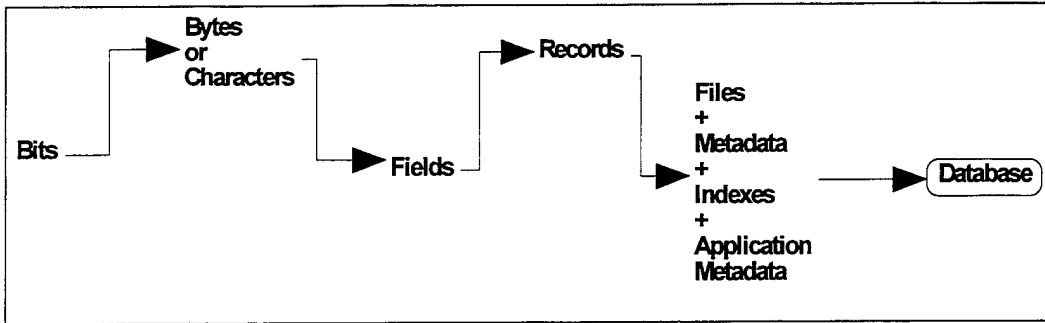


Figure 1: Hierarchy of data elements in database processing

3. Components of a Database Processing System

Figure 2 shows the main components of a database system. The *database* is processed by the *DBMS*, which is used by both *developers* and *users*. They can use the DBMS either directly or indirectly via *application programs*.

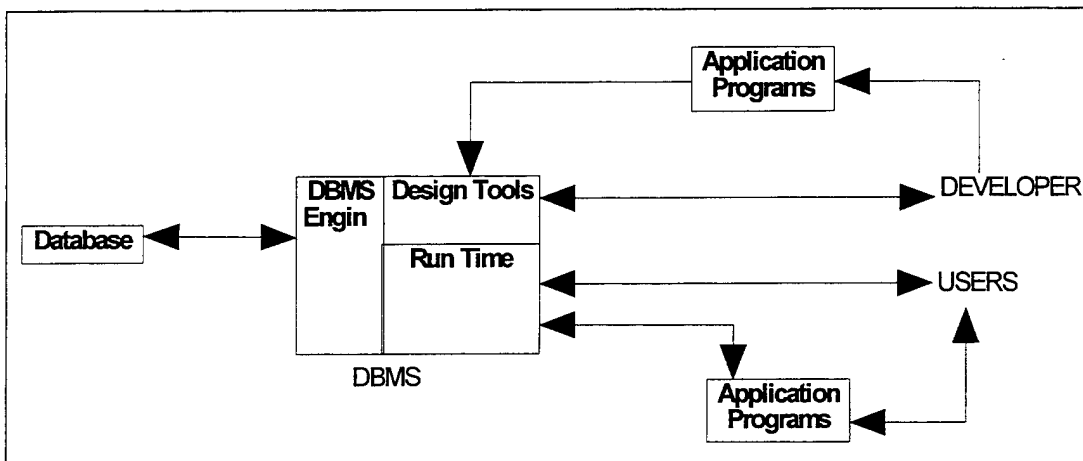


Figure 2: Components of a database system

B. DATABASE DEVELOPMENT METHODOLOGY

The database development methodology described here consists of four phases: definition, requirements, evaluation, and design. Each phase includes a number of tasks, as they are presented in Whitten [Ref. 3].

During the definition phase the tasks are to form the working team, define the problem, establish the scope, and assess feasibility issues.

During the requirements phase, the tasks are to create the user's data model, as well as the functional components of the application. This is accomplished by interviewing the users and by using prototypes to help determine user requirements.

During the evaluation phase, the tasks are to select the system's architecture and reassess feasibility issues.

During the design phase, the tasks are to develop the database design and the application design. The database design consists of structuring the relations and establishing relationships among them. The application design deals with the design of the menus, reports, and forms.

During the implementation phase, the tasks are to construct the database, build the application, and install it.

The requirements, design, and implementation phases are detailed in the following sections.

C. REQUIREMENTS ANALYSIS / SPECIFICATIONS

The first step in application development is to accurately obtain the system's information requirements from the potential users. No system can be designed without first understanding the current processes intended for improvement. After the system's definition and primary analysis phase, where the general goals of the system are determined, the requirements phase follows. The purpose of this phase is to determine, as specifically as possible, what the system must do. Many times, this is difficult to be accomplished because the users do not know what they really want. There are two tasks in this phase. The first task is to develop a user's data model and the second task is to determine the functional components of each application that will use the database.

1. Data Requirements

During the data requirements phase, the major goals are to build a data model that documents the "things" that are going to be represented in the database, determine the characteristics of those "things" that need to be stored, and determine the relationships among them. The user's data model describes the objects that must be stored in the database, along with their structure and the relationships that they have with one another. The output of the data requirements phase is a statement of requirements. This statement can take a variety of forms: a verbal description, an entity-relationship or objects diagrams, one or more prototypes, or any combination of the above.[Ref 1]

The "things" that are represented in the database are referred to as either entities or semantic objects (in some cases just objects) depending on the modeling technique that the designer follows. In this thesis the entity-relationship model will be followed.

2. Entity - Relationship Model

An entity is something that can be identified in the user's work environment; something important to the users of the system that is to be built [Ref. 1]. Entities are grouped into entity classes, or collections of entities of the same type. An entity class is the general form or description of a thing, such as PRODUCT, whereas an instance of an entity class is the representation of a particular entity, such as PRODUCT B1234. The terms entity and entity class are often used interchangeably. There are usually many instances of an entity in an entity class. For example, within the class PRODUCT, there are many instances - one for each product represented in the database.

Entities have attributes or properties which describe the entity's characteristics. The E-R model assumes that all instances of a given entity class have the same attributes. Entity instances have names that identify them. The identifier of an entity instance is one or more of its attributes.

Entities can be associated with one another by using relationships. The E-R model contains both relationship classes and relationship instances. Relationship classes are associations among entity classes, and relationship instances are associations among entity instances. Relationships can have attributes. A relationship can include many entities; the number of entities in a relationship is the *degree* of the relationship.

In entity-relationship diagrams, the entities are shown in rectangles, and the relationships are shown by the lines that connect the entities. The maximum and minimum number of entities that can participate in a relationship is also shown on the diagram. The maximum number of entities that can participate in a relationship, or *maximum cardinality*, is usually shown by using a crow foot (if it is many) or by a hash mark (if it is one). The *minimum cardinality*, minimum number of entities that can occur on one side of the relationship, is usually indicated by a hash mark (if it is one) or an oval (if it is zero).

In Figure 3 we can see an example of an E-R diagram. In this example, ENTITY No 1 can relate to ENTITY No 2 with a minimum of one instance and a maximum of many instances. ENTITY No 2 can relate to ENTITY No 1 with a minimum of zero instances and a maximum of many instances. ENTITY No 2 can relate to ENTITY No 3 with a minimum of zero instances and a maximum of many instances. ENTITY No 3 can relate to ENTITY No 4 with a minimum of one instance and a maximum of one instance. ENTITY No 4 can relate to ENTITY No 3 with a minimum of one instance and a maximum of one instance.

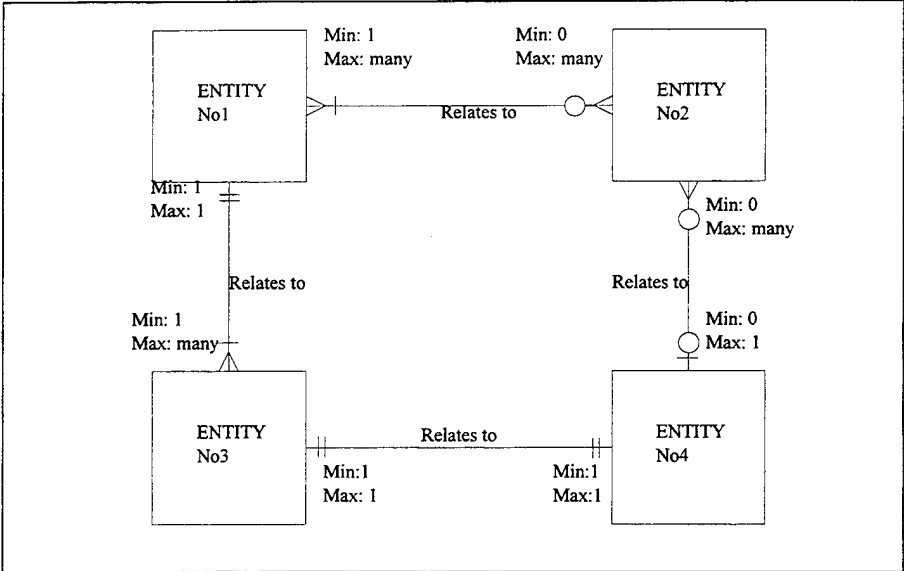


Figure 3: Example of an E-R diagram

After the data model has been developed, the designer should consider business rules that may restrict processing against entities. Business rules may or may not be enforced by the DBMS or by the application program. As Kroenke points out in [Ref. 1: p. 65], some business rules are written in manual procedures that the users of the database application are to follow. At this point, the way in which the rules are to be enforced is not important. What is important is to document these rules so that they become part of the system's requirements.

Databases do not model the real world, although it is a common misconception that they do. Rather, databases are models of the users' model of their business world. The appropriate criterion for judging a data model is whether the model fits the users' mental conception of their world.

3. Data Dictionary

A data dictionary (or project dictionary, as it is sometimes called) is a catalog of requirements and specifications for a new information system. [Ref. 3: p. 331] It provides definitions of all the data items in the database. During the definition phase, the analysts try to capture and store data about the system, and specify the inputs and outputs that the system will generate. These are represented with pictorial models such as data flow diagrams, entities, data stores, etc. The data dictionary expands this pictorial model and captures the detailed requirements for every input, output, and data store. The suggested approach for building the data dictionary should be in terms of "what" data are captured and not in terms of "how" data are formatted or presented.

4. Process Requirements

All application systems process data to produce information and maintain stored data. These requirements should be logically modeled. In order to implement processes as programs, a process model is needed. A process model is a picture of the flow of data through the system and the processing that must be performed on the data. These processes interact or interface with one another. These interactions take the form of data flows between processes, which is the reason that they are sometimes called data flow models. One of the most popular system modeling tools for capturing process requirements is the data flow diagrams (DFDs).

Data flow diagrams are very different from flow charts, in the following ways:

- Processes on a DFD can operate in parallel; several processes may be working simultaneously. This is a key advantage over flowcharts, which tend to show only sequences of processes.
- DFDs show the flow of data through a system unlike flowcharts that show steps in an algorithm.
- DFDs can show processes that have dramatically different timing while flowcharts cannot.

The following describes the basic components of a data flow diagram as they appear in Whitten [Ref. 3]. A sample DFD model is shown in Figure 4.

a. *Internal or External Entity*

Every system has a boundary. This boundary is defined by the internal or external entities that provide the net input to the system and receive the net output from the system. The entities sometimes are called sources or destinations, depending on whether they are inputs or outputs, respectively. Names and titles can be used to describe the label of the entities. Entities never interact directly with data stores, and relationships between entities are not modeled.

b. *Process*

The emphasis on any DFD is given to the processes, sometimes called activities. Processes transform inputs into outputs and transform the structure of data into information contained in the data. The logic or the procedure that a process uses to complete its task is not shown. Processes are titled by using verb-clause form.

c. *Data Store*

A data store, as the name implies, shows the logical storage of the information. Data flow from a data store represents the "usage" of data. This is the place where the data are stored after a process, or from where they are retrieved to be processed.

d. *Data Flow*

Data flows represent inputs or outputs that move from or to processes and from or to data stores. They are titled by a noun-clause form. Data transferred together

must be shown as a single data flow no matter how many documents are physically involved.

e. Leveling of Data Flow Diagrams

When studying, analyzing, and designing a system, it is good to have a generic pictorial outline of what the system does or will do when it is implemented. This pictorial outline, which is called "Decomposition Diagram", or "hierarchy chart", shows the top-down functional decomposition or structure of a system. Decomposition diagrams also provide an outline for drawing the DFDs. Only the processes are presented on decomposition diagrams, and they are connected to form a treelike structure. Process names conform with the ones that are referred to in the DFDs. The top process is called the root; it is exploded or factored out to subsystems, functions, or tasks. It defines the scope and boundary of the system to be developed.

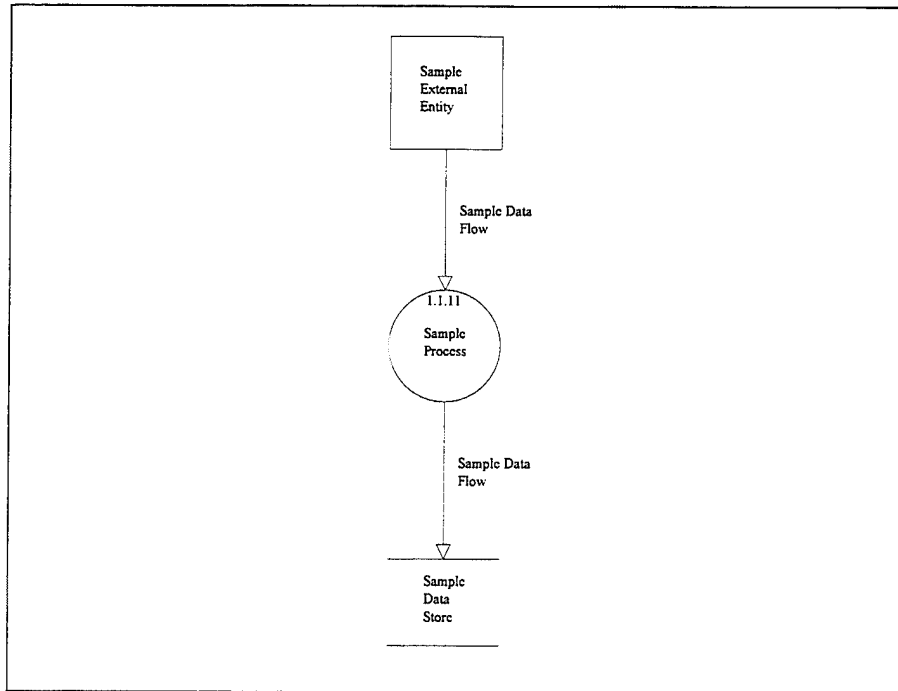


Figure 4: Sample DFD Model

D. DATABASE DESIGN

This part addresses the logical database design and the application design. [Ref. 2]

1. Logical Database Design

After the E-R model is developed, the next step is to transform the entities into a relational design. The relational model is important for two reasons. First, since the constructs of the relational model are general, it can be used to express DBMS-independent designs. Second, the relational model is the basis for an important category of DBMS products. Being familiar with this model helps implement databases using one of these products. [Ref. 1: p. 125]

a. Relational Model

A relation is a two-dimensional table. Each row, or tuple, in the table holds data that pertains to something or a portion of something in the user's environment. Each column, or attribute, of the table contains data regarding an attribute. For a table to be a relation, it must meet certain restrictions:

- The cells of the table must be single valued; neither repeating groups nor arrays are allowed as values.
- All of the entries in any column must be of the same kind.
- Each column has a unique name, and the order of the columns in the table is insignificant.
- No two tuples in a table may be identical, and the order of the tuples is insignificant.

Not all relations are equal. Some are better than others. Normalization is a process for converting a relation that has certain update problems to two or more relations that do not have these problems. Even more important, as Kroenke indicates [Ref. 1: p. 125], normalization can be used as a guideline for checking the desirability and correctness of relations.

b. Classes of Relations

Relations can be classified by the types of modification anomalies (deletion anomaly, insertion anomaly, and referential integrity constraint) to which they

are vulnerable. These classes of relations and the techniques for preventing anomalies are called *normal forms*. The normal forms are:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)
- Domain / Key Normal Form (DK/NF)

Each of the higher normal forms contains the lower ones. This means, for example, that a relation that is in the third normal form is also in both first and second normal forms. Therefore the steps in the normalization process are progressive, and one normal form follows another. In each step only certain anomalies are eliminated. It is mandatory for relational database designers to satisfy the requirements of all the normal forms to ensure that all anomalies have been eliminated, although in practice relations are usually normalized to the Third Normal Form.

2. Application Design

The design phase includes the design of both the database and the application. An application is the collection of menus, forms, reports, and programs that provide a means

to update, display, and control the objects of the data model. During the application design, the specific structure of forms, reports, menus, and query facilities are defined. Also, the logic of transaction programs is developed. The application design will be discussed further in Chapter IV.

E. DATABASE IMPLEMENTATION

The system's implementation is the set of activities following the logical design, and consists of the production of a working system that accepts inputs from the user, processes data, and produces the desired outputs. One very important task during the development of a software application is the development of the user's manual and the documentation of the development process.

III. REQUIREMENTS ANALYSIS FOR OADS

In this chapter we present both the data and the process requirements for the OADS application. We describe the data model and the corresponding data flow diagrams that represent the data flow that create, update, and display the entities of the data model.

A. DATA REQUIREMENTS

Data requirements are captured in the form of entities, attributes, and relationships, and the associated data dictionary. This application consists of ten entities. They are shown in the E-R diagram of Figure 5.

1. Entities

a. *Command Entity*

A command has a *Command Name*, which uniquely identifies it, is commanded by a *Commander Name*, who has a *Commander Rank*, and the base of the command is located in a *Base Location*. The command is organized into subcommands.

b. *Subcommand Entity*

A subcommand has a *Subcommand Name*, which uniquely identifies it, is commanded by a *Subcommander Name*, who has a *Subcommander Rank*, and the base of the subcommand is located in a *Base Location*. The subcommand controls a set of ships.

c. Ship Entity

A ship has a *Hull Number*, which uniquely identifies it, and a *Name*, belongs to a specific *Type* of ships, and has a *Number of personnel* in it. The ship is commanded by a *CO_Name*, who has a *CO_Rank*. Every ship belongs to a specific subcommand.

d. Port Entity

Each port has a Port Number, which uniquely identifies it, and a *Port Name*, and it is located in a general geographical *Location*. Each port may has *Watering*, *Fueling*, and *Maintenance Capabilities*.

e. Exercise Entity

An exercise has an *Exercise Number*, which uniquely identifies it , and an *Exercise Name*. The date that the exercise begins is the *Date begins* and the date that the exercise finishes is the *Date ends*. Each exercise has a *Geographical Location* where the exercise took place. The exercise consists of a number of drills.

f. Drill Entity

A drill has a *Drill Number*, which uniquely identifies it , and it belongs to a specific *Drill Type*. It is performed in a *Drill Date* at a specific *Time begins*, and it finishes at a specific *Time ends*. Also, each drill describes a specific *Objective*.

g. Ship-Port Entity

Each ship that has visited a port has gone to that port at a *Date begins*, and has remained there for a number of *Hours*. The ship, while in port, was in a specific *Ship's state* (maintenance or readiness) and may have taken an *Amount of Fuel in lt.*

h. Command-Exercise Entity

Each exercise was performed by a command for some *Hours*.

i. Subcommand-Drill Entity

A subcommand may performed a drill for some *Hours*.

j. Ship-Drill Entity

A ship may participated to a drill for some *Hours*. During that drill, the ship may have some *Torpedoes used*, *A/A Missiles used* or *A/S Missiles used*. Also, the ship may detected a number of *Submarines detected*.

2. Relationships

The E-R diagram contains contains eleven one to many relationships. These relationships and their cardinalities are shown in Figure 5.

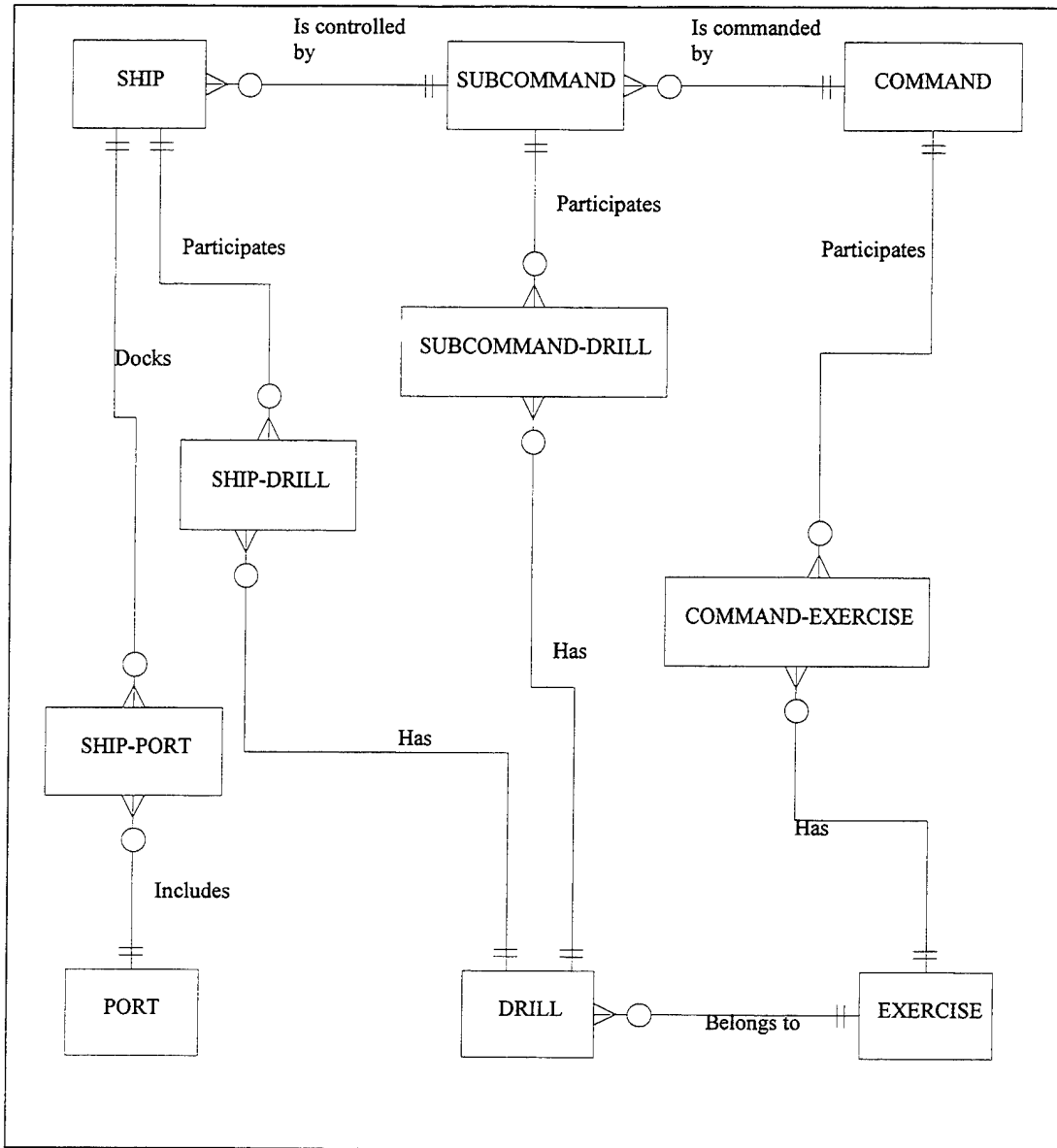


Figure 5: Application's E-R Diagram

B. DATA DICTIONARY

The OADS application data dictionary is shown in Appendix A. It describes each entity, each attribute in the entities, the data type, and definition of each attribute.

C. PROCESS REQUIREMENTS

In this application, the decomposition diagram and the data flow diagrams that describe the system's functionality are shown in Appendix B. The system has four levels. The zero level is the overall system picture named *Operational Activity System*. It is factored out into two different subsystems, *Update Subsystem* and *Retrieval Subsystem*. The system's hierarchical outline form is shown in Figure 6.

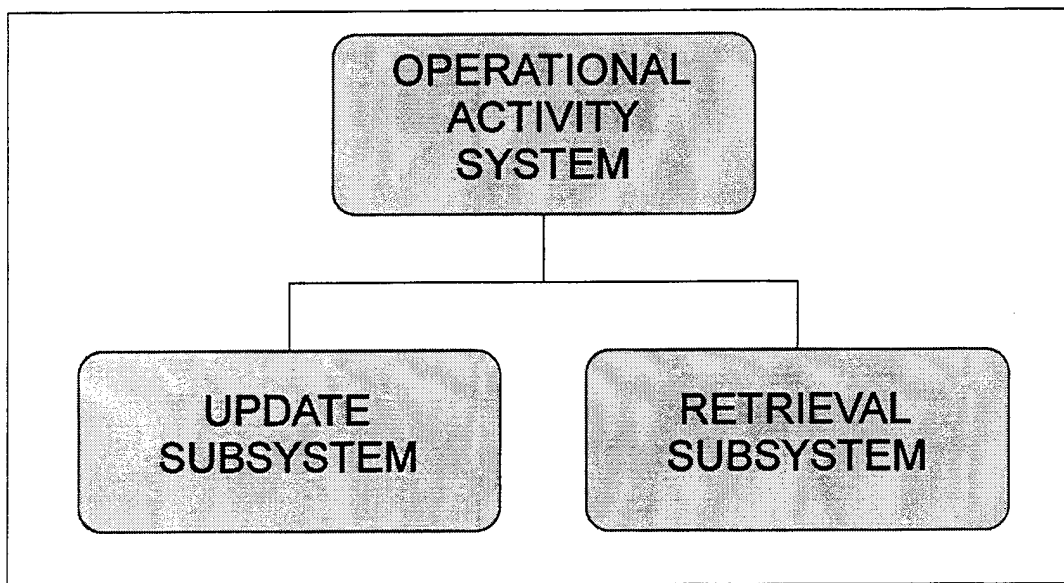


Figure 6: Application's Process Outline

1. Update Subsystem

This subsystem has ten processes: *Update Command Process*, *Update Subcommand Process*, *Update Ship Process*, *Update Exercise Process*, *Update Drill Process*, *Update Port Process*, *Update Exercise Per Command Process*, *Update Drill Per*

Subcommand Process, Update Drill Per Ship Process, and Update Port Per Ship Process. Each of these processes consists of three subprocesses: *Addition Process, Deletion Process, and Modification Process.* Update Subsystem's process hierarchy is shown in Figure 7.

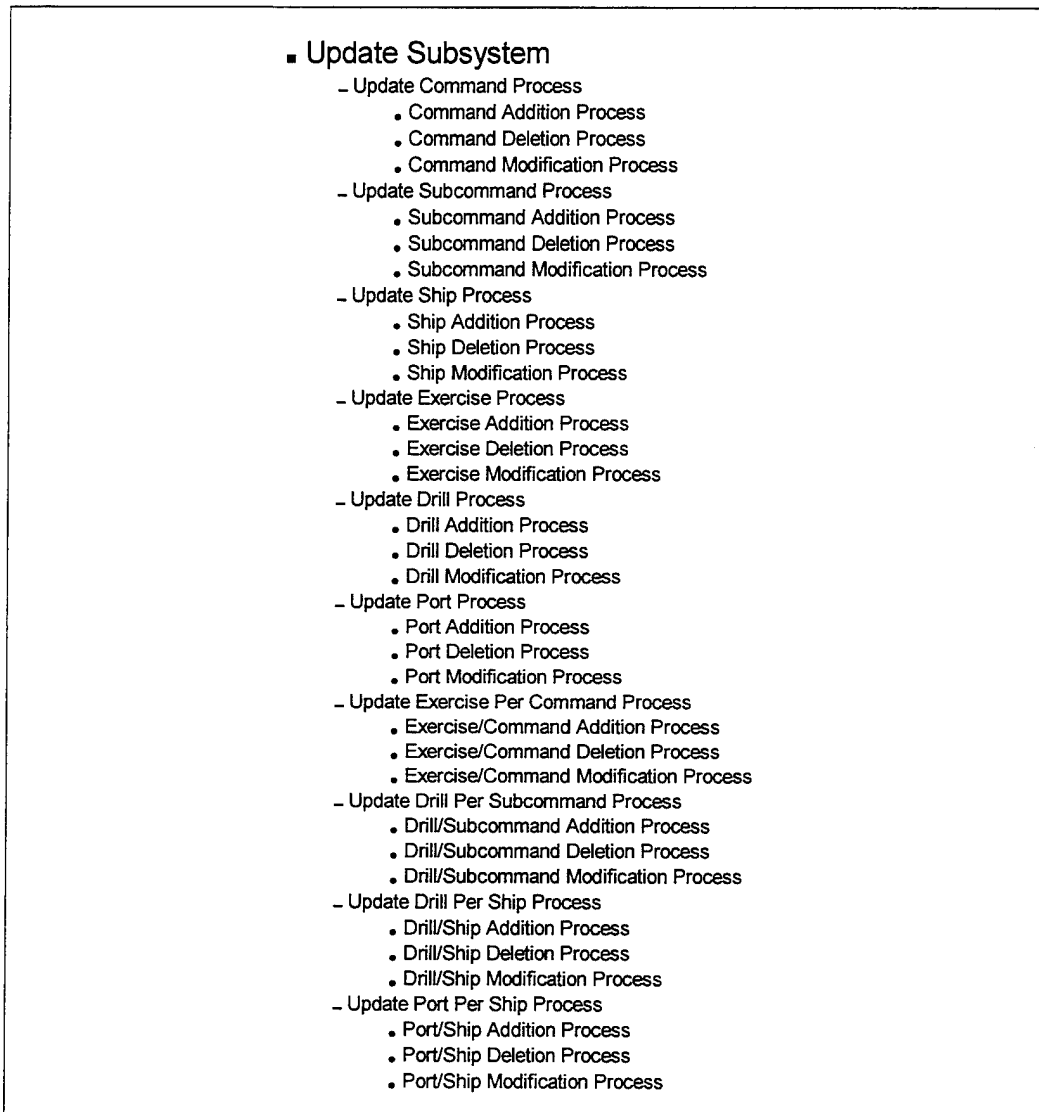


Figure 7: Update Subsystem

2. Retrieval Subsystem

The retrieval subsystem has three processes: *Report Retrieval Process*, *Record Retrieval Process*, and *Query Retrieval Process*. The *Report Retrieval Process* consists of five subprocesses: *Process Exercises Per Command*, *Process Drills Per Exercise*, *Process Ships Per Subcommand*, *Process Drills Per Ship*, and *Process Ships Per Port*. The *Record Retrieval Process* consists of five subprocesses: *Process Drills Per Exercise*, *Process Subcommands Per Command*, *Process Drills Per Ship*, *Process Activity Hours*, and *Process Ships Per Port*. The *Query Retrieval Process* consists of four subprocesses: *Process Activity Hours Queries*, *Process Drill Queries*, *Process Organization Queries*, and *Process Ship Activity Queries*. The user in the General Staff of the Hellenic Navy will be able to produce any report that will help the General Staff in preparing the operational activity schedule for commands, subcommands, and ships. Any query can be performed to the existing data and the query results can be displayed in the user's desired format. Retrieval Subsystem's process hierarchy is shown in Figure 8.

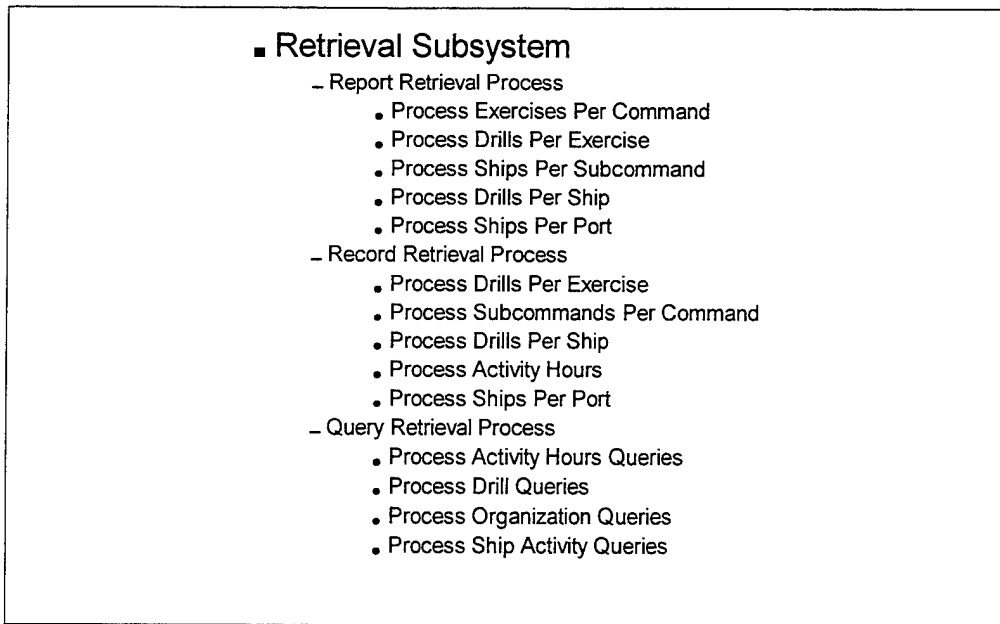


Figure 8: Retrieval Subsystem

D. **HARDWARE REQUIREMENTS**

The system being developed will be implemented on an IBM compatible PC platform found on many sites in the General Staff of the Hellenic Navy. The minimum hardware configuration is a 386 SX (16 bit architecture) processor running Windows 3.1 at 50 MHz, with 16 Mb of RAM (32 Mb recommended) and 540 Mb of hard drive. Also, a mouse or other Windows pointing device is required, in order to effectively utilize the capabilities of the system.

IV. LOGICAL DATABASE AND APPLICATION DESIGN FOR OADS

In this chapter we discuss the logical database and application design for OADS. In logical database design, the E-R model developed in the previous chapter is transformed into a relational schema in preparation for implementation using a specific DBMS. In application design, the data flow diagrams are used as a basis for developing the menus, forms, and reports for OADS. [Ref. 2]

A. LOGICAL DATABASE DESIGN

The ten entities, describing the user's environment, are transformed into ten relations. Relationships are presented using foreign keys; the primary keys are underlined and the foreign keys are indicated in italics. The complete relational diagram is shown in Appendix C.

1. Command Relation

This relation contains information about a command. It is derived from the COMMAND entity. Its primary key is Command Name. Other attributes are *Base Location*, *Commander Name*, and *Commander Rank*. It has a 1:M mandatory relationship to Subcommand relation, and a 1:M relationship to the Command-Exercise relation.

2. Subcommand Relation

This relation contains information about a subcommand. It is derived from the SUBCOMMAND entity. Its primary key is Subcommand Name. Other attributes are Command Name (foreign key), Subcommander Name, Subcommander Rank and Base Location. The Subcommand relation has a M:1 relationship with the Command relation, a 1:M relationship with the Ship relation, and a 1:M relationship with the Subcommand-Drill relation.

3. Ship Relation

This relation contains information about a ship. It is derived from the SHIP entity and its primary key is Hull Number. Other attributes are Name, Subcommand Name (foreign key), Type, CO_Name, CO_Rank and Number of personnel. It has a M:1 relationship to Subcommand relation, a 1:M relationship to Ship-Port relation, and a 1:M relationship to Ship-Drill relation.

4. Exercise Relation

This relation contains information about an exercise. It is derived from the EXERCISE entity. Its primary key is Exercise Number. Other attributes are Exercise Name, Date begins, Date ends and Geogr. Location. It has a 1:M mandatory relationship to Drill relation and a 1:M relationship to Command-Exercise relation.

5. Drill Relation

This relation contains information about a drill. It is derived from the DRILL entity. Its primary key is Drill Number. Other attributes are Exercise Number (foreign key), Drill Date, Time begins, Time ends, Drill Type and Objective. It has a 1:M relationship to Ship-Drill relation, a M:1 relationship to Exercise relation, and a 1:M relationship to Subcommand-Drill relation.

6. Port Relation

This relation contains information about a port. It is derived from the PORT entity. Its primary key is Port Number. Other attributes are Port Name, Location, Watering Capability, Fueling Capability and Maintenance Capability. It has a 1:M relationship to Ship-Port relation.

7. Ship-Port Relation

This relation contains information about a ship and its visit to a port. It is an intersection relation that represents the many to many relationship between the SHIP and PORT entities. Its primary key is a composite one and it consists of the attributes: Hull Number, Port Number and Date begins. Other attributes in this relation are Amount of Fuel in lt., Hours, and Ship's state. It has a M:1 relationship to Port relation and a M:1 relationship to Ship relation.

8. Subcommand-Drill Relation

This relation contains information about a subcommand and the drills that the subcommand has performed. Alternatively one can say that this relation contains information about a drill and the subcommands that participated in it. The primary key of this relation is a composite one and it consists of the attributes Drill Number and Subcommand Name. Other attribute of this relation is Hours. It has a M:1 relationship to Drill relation and a M:1 relationship to Subcommand relation.

9. Command-Exercise Relation

This relation contains information about a command and the exercises that the command has performed. Alternatively one can say that this relation contains information about an exercise and the commands that participated to it. The primary key of this relation is a composite one and it consists of the attributes Exercise Number and Command Name. Other attribute of this relation is Hours. It has a M:1 relationship to Exercise relation and a M:1 relationship to Command relation.

10. Ship-Drill Relation

This relation contains information about a ship and the drills that the ship has performed. Alternatively one can say that this relation contains information about a drill and the ships that participated to it. The primary key of this relation is a composite one consisting of the attributes Drill Number and Hull Number. Other attributes of this relation are Hours, Torpedoes used, A/A Missiles used, A/S Missiles used, and

Submarines detected. It has a M:1 relationship to Drill relation and a M:1 relationship to Ship relation.

B. APPLICATION DESIGN

In application design, the data flow diagrams developed in the requirements phase are used as the basis for designing the system's menus, forms, queries, and reports. The following section provides a brief explanation of each.

1. Menus

OADS is a menu-driven application. The reason for using menus is because they are self explanatory and are therefore easy to use by the Hellenic Navy General Staff users. The menu structure of OADS follows closely the decomposition diagram developed during process requirements.

2. Forms

Forms are the user's primary interface with the database. They are used for entering, modifying, and displaying data retrieved from the database. Special care was paid in designing the forms for OADS. Every effort was made in designing them to be easy to use and less prone to errors.

3. Queries

Queries results are an output of the system. The user can choose from a predetermined set of queries and the results can be sent to the screen, or to the printer. In

designing of the set of queries, every effort was made to be close to the queries that are currently asked.

4. Reports

Reports are the main output that the system generates for distribution to a variety of users. Reports can be sent to the screen, to a file, or to the printer. Similar to designing forms, special care was paid in designing the reports for OADS. Every effort was made in designing them to be natural, logical, close to the format that is currently in use and less prone to misinterpretation.

V. IMPLEMENTATION FOR OADS

In this chapter we will discuss the implementation of the OADS application and the construction of the database, as well as the installation of both the database and the OADS application. The Paradox 5.0 database management system for windows is introduced and used as the DBMS of choice for OADS implementation.

As it is indicated to Paradox User's Guide [Ref. 4], Paradox is a fast, full-featured, and easy to use relational database program designed to meet data management needs. Paradox can be used by computer users with all levels of experience from beginning database users to advanced developers. Paradox can be used either on a single computer (standalone) in the Hellenic Navy General Staff or on a Local Area Network (LAN). To use Paradox 5.0 on a stand-alone computer, you will need at a minimum:

- A 100% IBM compatible, protected mode capable 80386 or higher personal computer with a hard disk and a floppy drive.
- 6 MB RAM (8 MB is recommended.) Performance will increase with more memory.
- At least 20 MB of free hard disk space.
- EGA or higher video monitor.
- Microsoft Windows version 3.1 or later.
- Although not required, a mouse is strongly recommended.

The user interface for Paradox supports multiple windows, pull-down menus, speed bars, dialog boxes, and other graphical user interface components. Also, Paradox provides mouse support. For instance, you can change directories when loading tables by pointing and clicking at a directory tree with the mouse. Paradox's Query By Example (QBE) capability is one of the product's strong features. Complex queries can be run against single or joined tables, and query images can be saved for later use. A variety of exact and inexact queries can be performed, and there is support for wildcards, data ranges, pattern searches, and logical conditions. Phonetic searches can be done with Paradox's "Like" operator.

Paradox 5.0 has ObjectPAL, which is a high-level, event-driven, object-based, visual programming language. You can use ObjectPAL to create a completely customized application, one with entirely new buttons, menus, dialog boxes, prompts, warnings, and help. ObjectPAL is the user's tool for creating the user interface for a database application.

A. DATA IMPLEMENTATION

In data implementation, the relations and their attributes developed during logical database design are transformed into tables and data fields, respectively. Table structures are created in Paradox 5.0 by choosing File/New/Table, or right clicking the Open Table Toolbar button and choosing New. The structure of the new empty table, which matches the corresponding relation developed during the design phase, is then specified. For each

field of the table, its name, field type, and whether it is a key field are entered. The data types supported by Paradox 5.0 and their abbreviations are:

- Alpha (A), for alphanumeric fields up to 255 characters in length.
- Number (N).
- Money (\$), for currency amounts.
- Memo (M), for a memo up to 240 characters in table view.
- Date (D).
- Time (T).
- Timestamp (@), for both a date and time value.
- Graphic (G), for pictures in .BMP format.
- Logical (L), for values representing true or false.

Once the definition of a table is completed, a user can enter values in the table directly or through a form.

B. APPLICATION IMPLEMENTATION

In general, Paradox 5.0 applications are built in the following stages:

- *Get your data together.* Build and populate tables.

- *Create the forms.* Although you can write scripts and store code in libraries, the vast majority of ObjectPAL code is attached to objects in forms. The best way to start is to create the forms, place the objects, and run the forms.
- *Attach ObjectPAL code to the object's built-in methods.* Modify an object's behavior by attaching code to the appropriate built-in method. Built-in methods execute in response to events, so to select the appropriate built-in method, you should determine what the object should do and when it should do it.
- *Test and refine the application.* Paradox makes it easy to develop an application one piece at a time. You do not have to code the entire application before you can run a form and make sure things are happening as you expect.

The above procedure was followed in the implementation of the OADS application. First, the tables were built and populated, the forms were created, system's reports were produced, and specific queries were performed. Next, the menu hierarchy was built, and ObjectPAL code was attached. The system was tested one form at a time, and system's code was debugged. After implementation, the system was tested as a whole.

The next chapter discusses other issues that need to be taken into consideration before OADS can be fully operational.

VI. OTHER ISSUES

This chapter discusses important issues concerning the development of the "OADS" application, such as security, training, conversion, maintenance, and future upgrades. The issues that this chapter presents are the same issues that have been covered in Tsongas' thesis [Ref. 2], but customized for the OADS application.

A. SECURITY

Paradox 5.0 offers a very flexible password scheme with table-by-table, script-by-script, and option-by-option password protection. Access to a given table can be limited on a field-by-field basis.

OADS users will have some access control authorities depending on their rank, their name, and their duties in the General Staff. The database administrator of the General Staff is responsible for assigning these duties and for determining each user's access control.

OADS physical security falls under the Navy's policy and procedures that enforce rules and activities for the General Staff's physical security. Moreover, the General Staff has to take proper measures to protect the hardware resources as well as the software and the applications.

B. TRAINING

One of the most important aspects of the implementation phase is the training plan. The training plan is designed to ensure that every user of the system knows the system's basic functions and how to perform them. The success of any information system depends on the skills of the operators. In the OADS system, the operators are officers, petty officers, and sailors who are familiar with the operational activities of commands, subcommands, and ships, currently run the system manually; and who need to be taught how the new system operates. The system itself is designed to be friendly and easy to use, and does not require the operator to have any advanced interpersonal skills. However, matching basic human characteristics and skills with a job's requirements is essential, especially when an automated system is to replace a manual old one.

The designer's proposal for the training is to start with the main users of the system and train them in the system's environment as well as its functions and operations. The main users of the system are defined as personnel working in the General Staff Operational Activity Schedule office. They are led by a Captain whose team normally consists of a Lieutenant Commander, two Lieutenants, three Petty Officers, and three sailors. As soon as the trained core learn how to operate the system, training for the rest of the personnel of the General Staff can be held.

C. CONVERSION

The future success of the new system depends on how well and quickly it is accepted by the users. With OADS application, it is hoped that the system is rather small and the users are already familiar with the environment; proper training will minimize these problems. Another way to minimize implementation problems is to select the correct conversion strategy.

1. Conversion Strategy

For the OADS application conversion strategy, the phased approach is proposed. The new system is easy to be implemented as soon as the data for a specific period of time have been collected. The transition from the old system to the new one is expected to last two months at most. With the phased conversion approach, the Hellenic Navy will have the following advantages:

- Elimination of risk involved with implementing the new system.
- Demands for extra manpower can be controlled.
- Shift from the old system to the new one is quite smooth.
- User resistance is minimized.

The period of three months is a reasonable interval for checking all the system's reports and making the users familiar with the OADS.

D. MAINTENANCE

Once the system passes the acceptance test, it is ready for delivery. Any modifications or enhancements after delivery is called maintenance. Attention should be paid to the fact that after a few years of operating the original system, maintenance becomes extremely tedious, error-prone, and expensive. In this case, management should recognize the problem and do a feasibility study on replacing the old system with a new one.

E. FUTURE ENHANCEMENTS

The OADS system design offers a flexible way for future upgrades. Paradox itself can be distributed on a network (LAN in the Hellenic Navy General Staff, or WAN between the Hellenic Navy commands) and allow applications to be shared among different users. The OADS system is able to produce all the designed reports in file format. This facility allows General Staff to electronically transfer their reports as soon as all the commands are connected to a common network.

VII. CONCLUSIONS AND LESSONS LEARNED

This thesis presented the design, development, and implementation of the Operational Activity Database System (OADS) application on a standalone computer in the Hellenic Navy General Staff. The OADS system will provide the General Staff with an automated system to create its primary operational activity schedules. OADS supports this mission by keeping track of all the operational activity records, maintaining them, producing reports, and providing the General Staff with ad hoc information. [Ref. 2]

No automated system is currently in use in the General Staff and all the operational activity schedules are prepared by a manual filing system, which is slow, inaccurate, and prone to errors. The main goal of developing the system is to increase effectiveness, efficiency, and accuracy of operational activity management. After OADS implementation, it is hoped that most of the current problems will be eliminated, and future enhancements will result in even greater efficiency in preparing the operational activity schedules.

System analysis and design tools and techniques were used to develop the system by modeling the user data and process requirements. Paradox for windows 5.0 was used as the database management system for the implementation because it is not only powerful but also meets the system's developmental requirements. The menu-driven environment is easy to use and quick to learn. The user friendly environment will hopefully eliminate the cultural resistance of the user that will result from the requirement to switch from the manual system to an automated one.

It is hoped that this thesis will be the motivation for other efforts to develop new systems and expand or update existing ones, in the Hellenic Navy. [Ref. 2]

Author's expectation is that the Hellenic Navy General Staff will adopt the OADS and utilize it in its maximum capabilities in order to effectively prepare the Operational Activity Schedule of the Hellenic Navy units. First of all, senior Officers of the General Staff must be convinced for the benefits of OADS and the easy of its use.

The main idea of this thesis and the thesis of Tsongas [Ref. 2] is to offer to the Hellenic Navy a set of applications that will ease the administrative, operational, and tactical functions. Hoping that the future Greek Officers, who are going to attend the ITM Curriculum in NPGS, will adopt this idea, we can make our Navy a technologically updated and "ready to go".

APPENDIX A: DATA DICTIONARY

<u>ELEMENT</u>	<u>TYPE</u>	<u>WIDTH</u>	<u>DESCRIPTION</u>
COMMAND			
Command Name	Alpha	20	Command's name.
Commander Name	Alpha	30	Commander's name.
Commander Rank	Alpha	20	Commander's rank.
Base Location	Alpha	30	Command's base location.
SUBCOMMAND			
Subcommand Name	Alpha	10	Subcommand's name.
Subcommander Name	Alpha	30	Subcommander's name.
Subcommander Rank	Alpha	20	Subcommander's rank.
Base Location	Alpha	20	Subcommand's base location.
EXERCISE			
Exercise No	Number		Exercise's #.
Exercise Name	Alpha	20	Exercise's name.
Date begins	Date		Date the exercise begins.
Data ends	Date		Date the exercise ends.
Geogr. Location	Alpha	40	Exercise's location.
DRILL			
Drill No	Number		Drill's #.
Drill Type	Alpha	10	Drill's type.

Drill Date	Date		Date drill begins.
Time begins	Time		Time drill begins.
Time ends	Time		Time drill ends.
Objective	Alpha	60	Drill's objective.

SHIP

Hull No	Alpha	5	Ship's Hull #.
Name	Alpha	20	Ship's name.
Type	Alpha	20	Ship's type.
CO_Name	Alpha	30	CO's name.
CO_Rank	Alpha	20	CO's rank.
Number of personnel	Number		Number of ship's personnel.

PORT

Port No	Number		Port's #.
Port Name	Alpha	30	Port's name.
Location	Alpha	30	Port's location.
Watering Capab.	Logical{Yes,No}		If port has watering capability.
Fueling Capab.	Logical{Yes,No}		If port has fueling capability.
Maintenance Capab.	Logical{Yes,No}		If port has any maintenance capability.

SHIP-PORT

Hull No	Alpha	5	Ship's Hull #.
Port No	Number		Port's #.
Date begins	Date		Date ship arrived to port.

Hours	Number		Hours ship remained to port in the same condition.
Ship's State	Alpha	20	The condition of the ship.
Amount of Fuel in It ship	Number		The amount of fuel that a received in a port.

COMMAND-EXERCISE

Exercise No	Number		Exercise's #.
Command Name	Alpha	20	Command's name.
Hours	Number		Hours that the Command participated in the Exercise.

SUBCOMMAND-DRILL

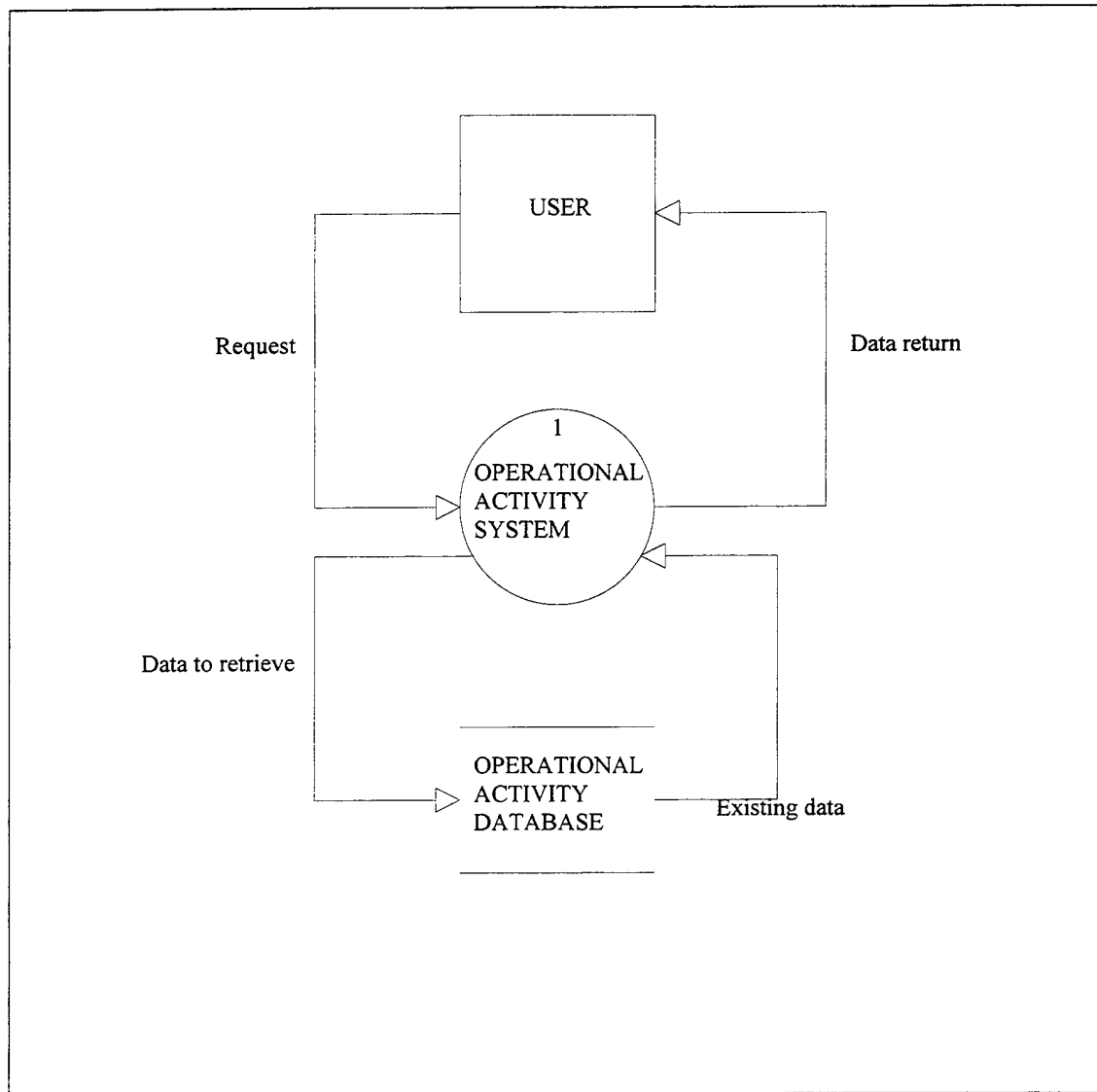
Subcommand Name	Alpha	10	Subcommand's name.
Drill No	Number		Drill's #.
Hours	Number		Hours that the Subcommand participated in the drill.

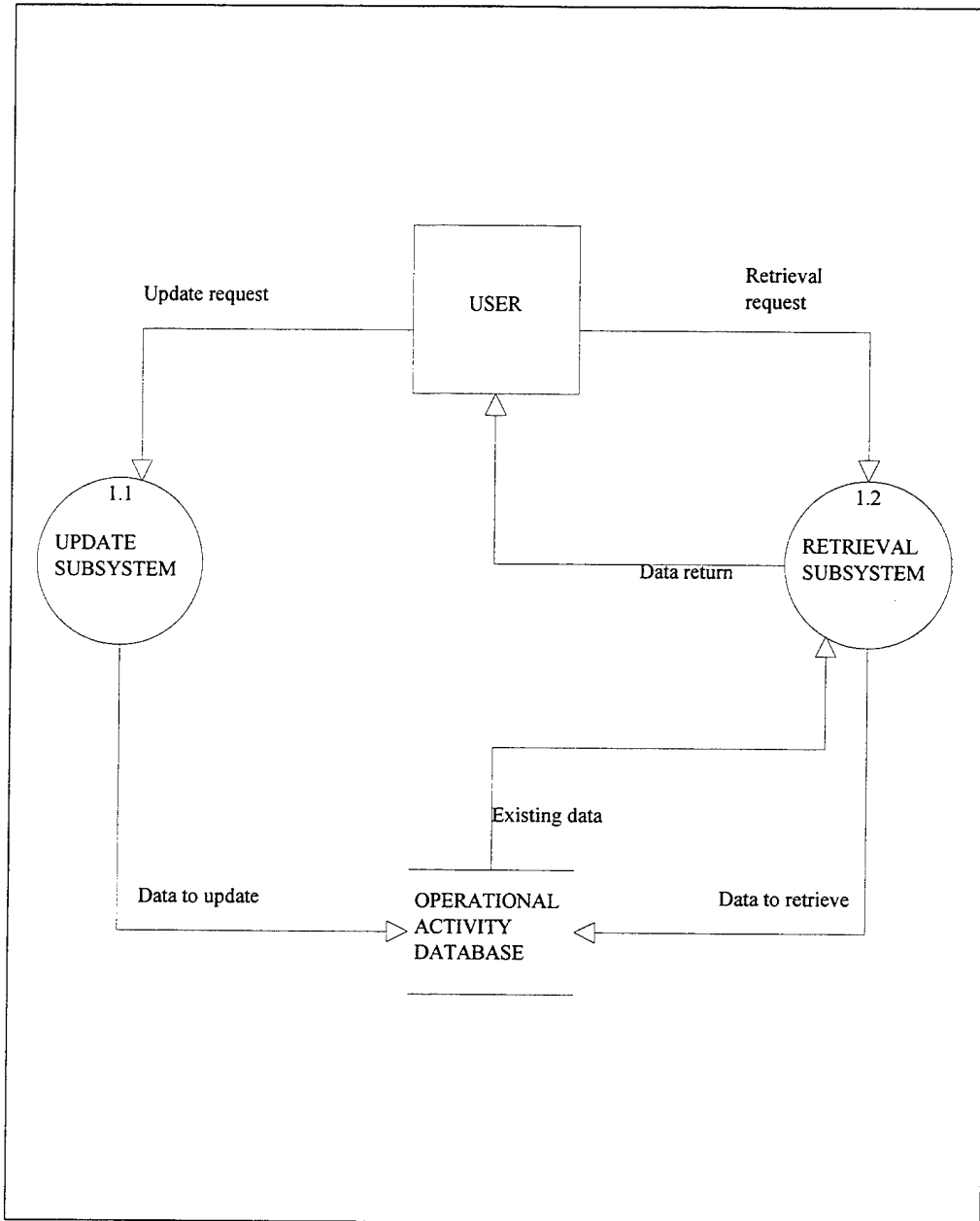
SHIP-DRILL

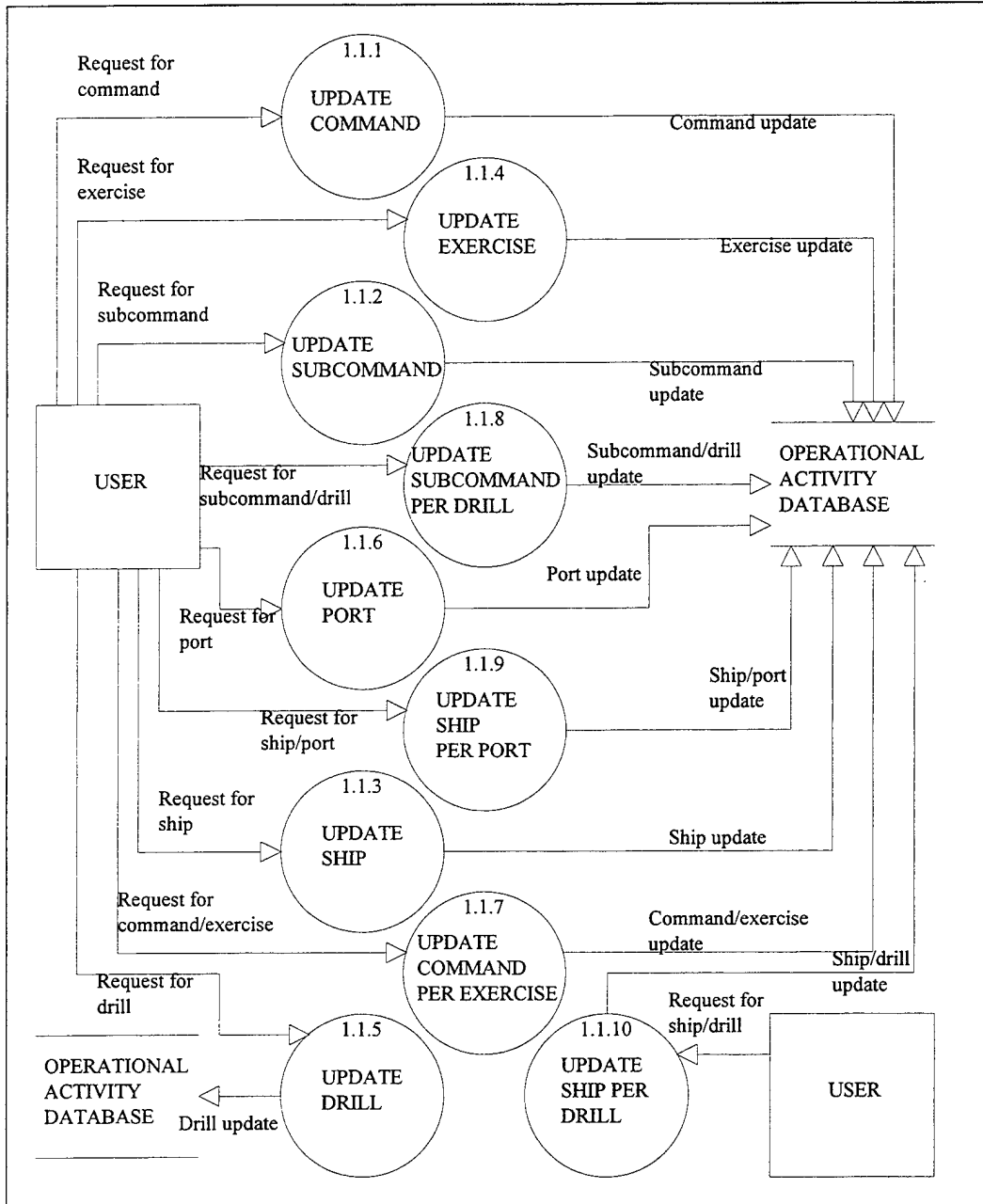
Drill No	Number		Drill's #.
Hull No	Alpha	5	Ship's Hull #.
Hours	Number		Hours that the Ship participated in the drill.
Torpedoes used	Number		Number of torpedoes used by the ship during the drill.
A/A Missiles used	Number		Number of A/A missiles used by the ship during the drill.

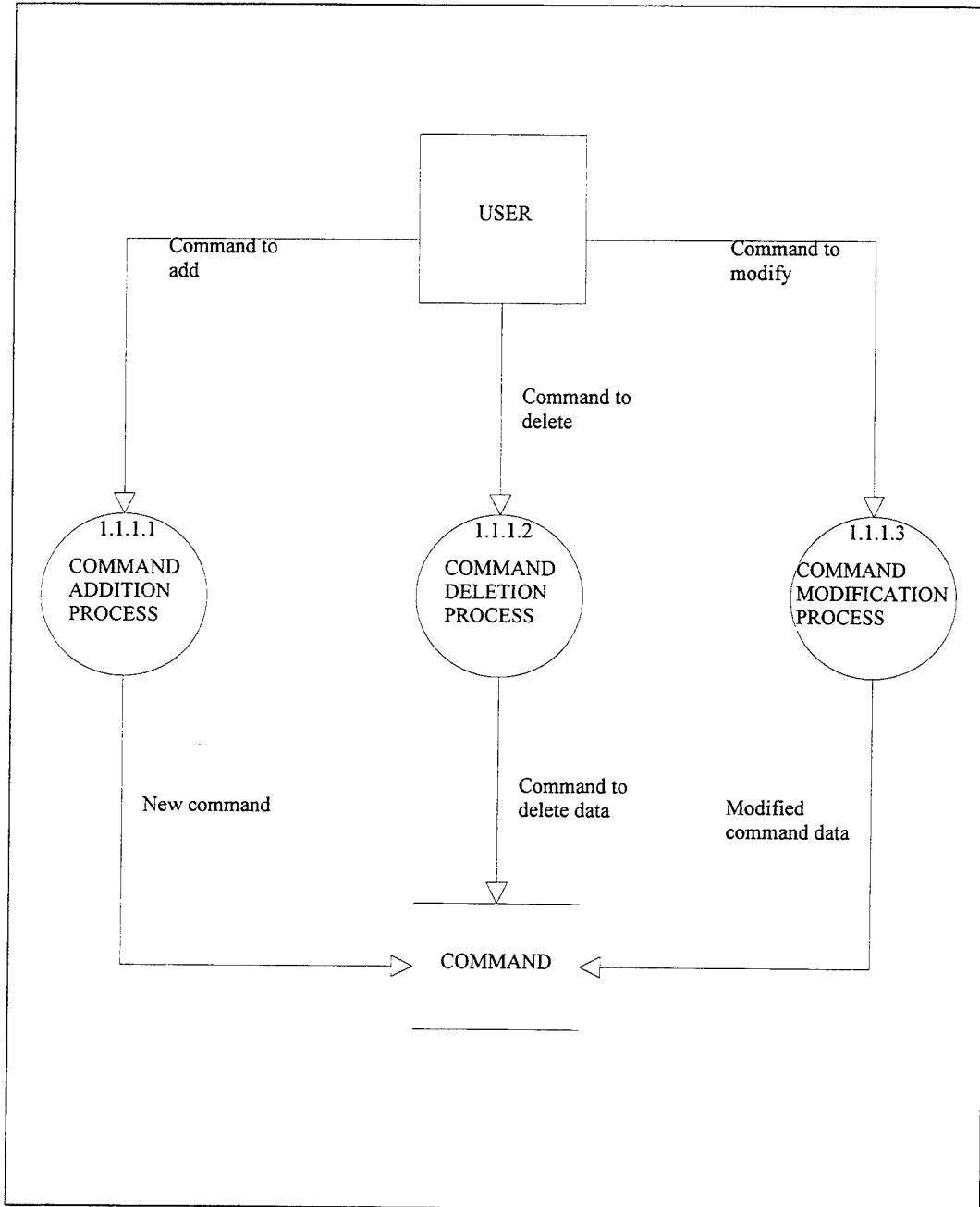
A/S Missiles used	Number	Number of A/S missiles used by the ship during the drill.
Submarines detected	Number	Number of submarines detected by the ship during the drill.

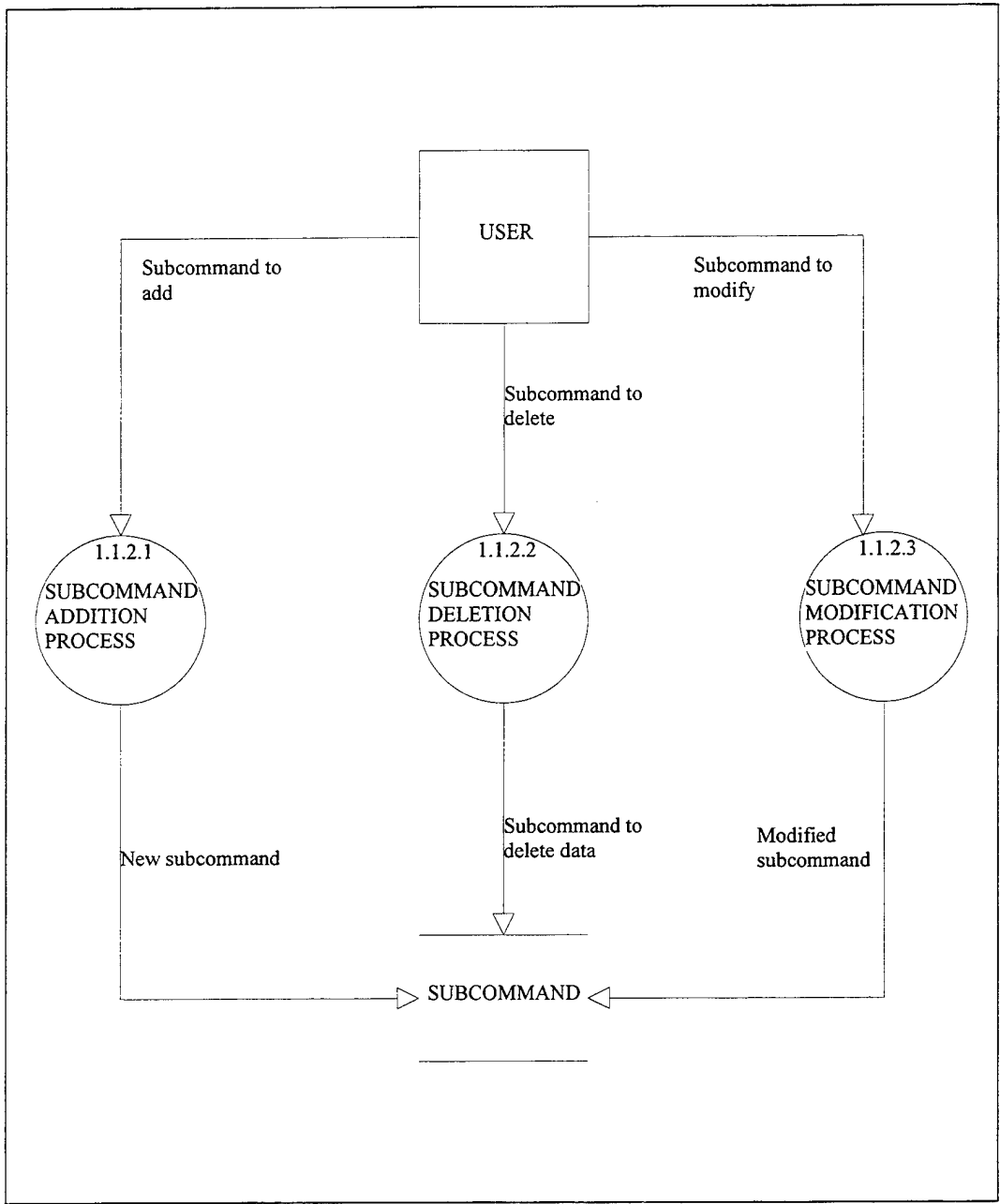
APPENDIX B: DATA FLOW DIAGRAMS

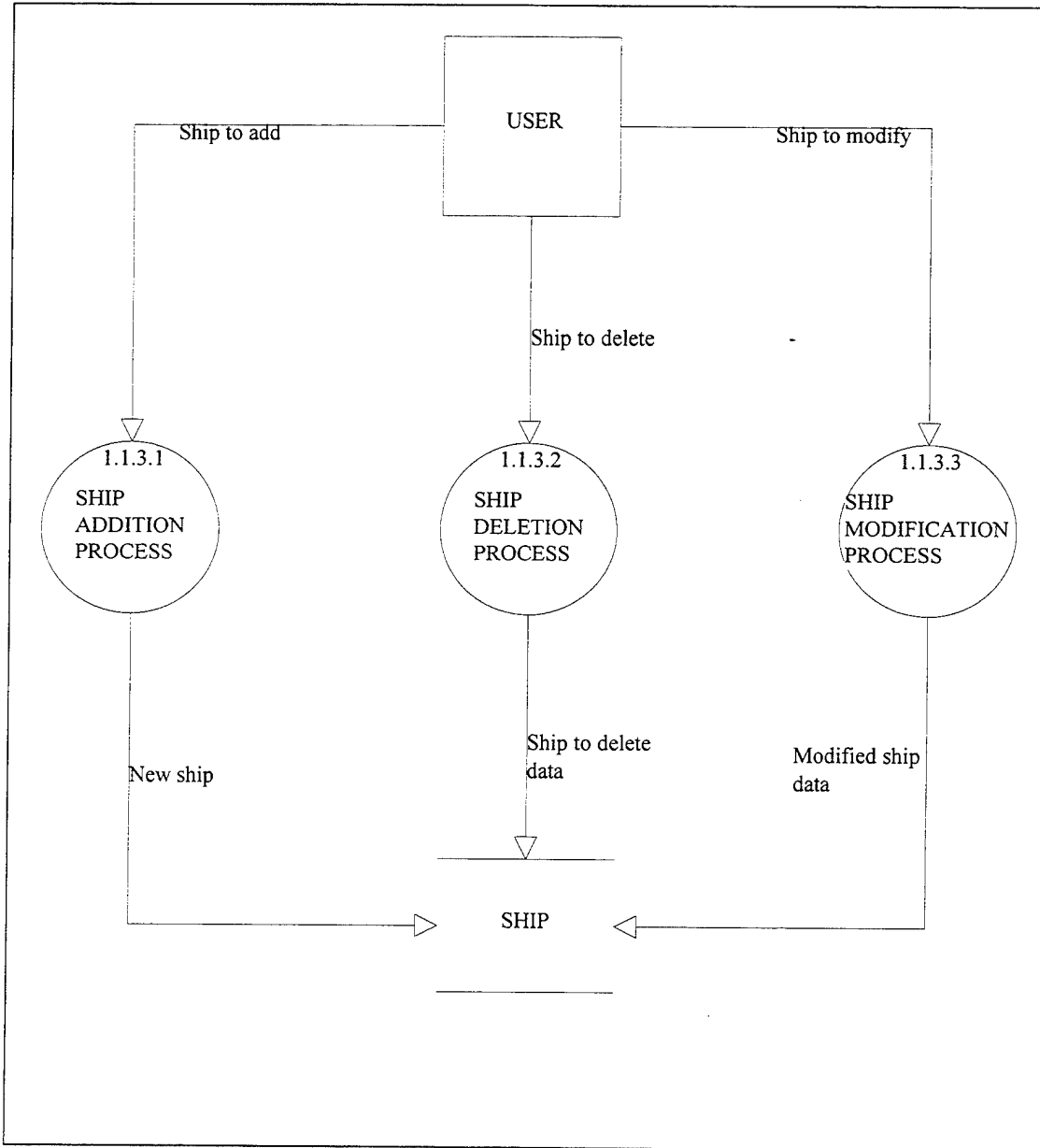


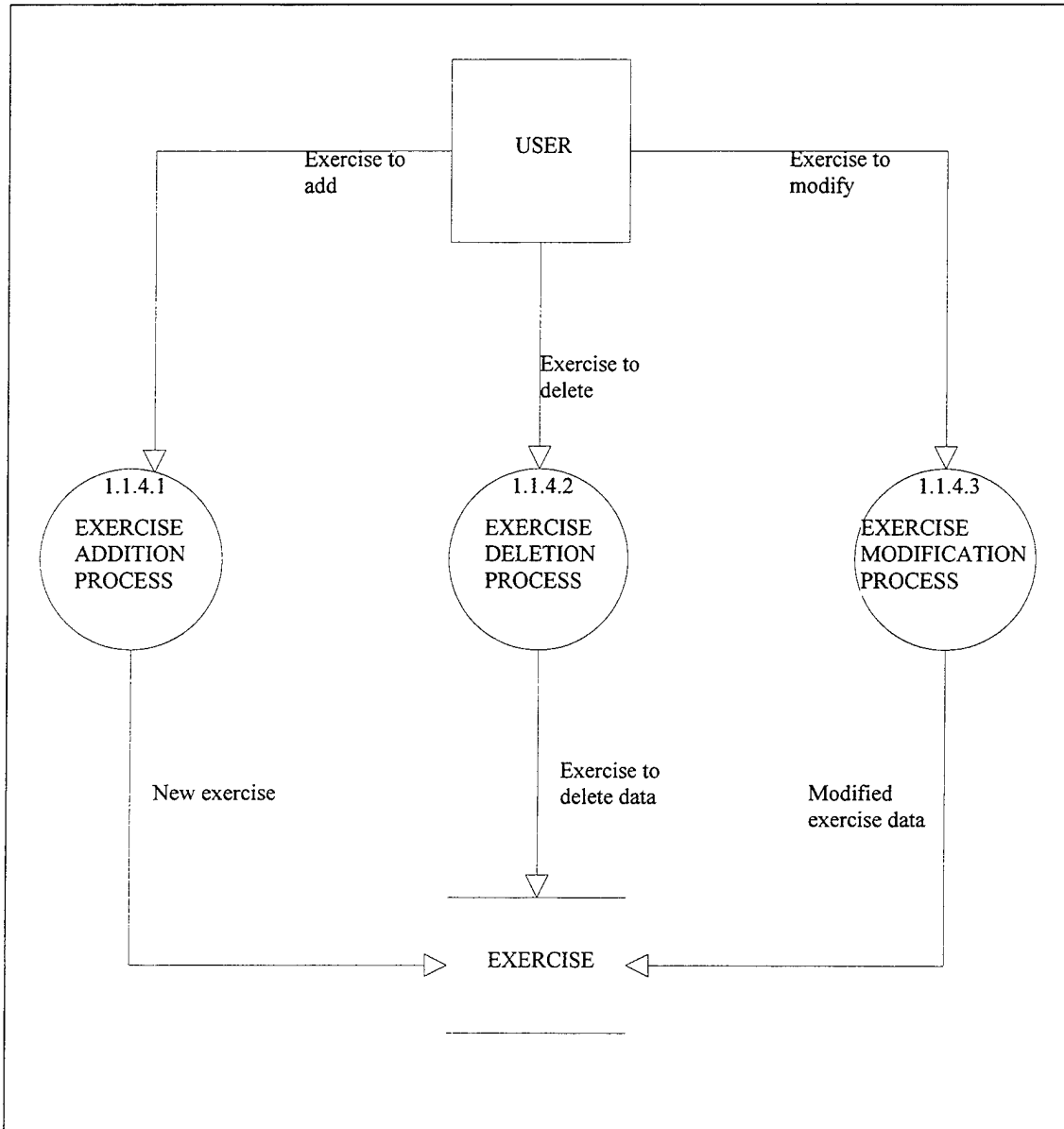


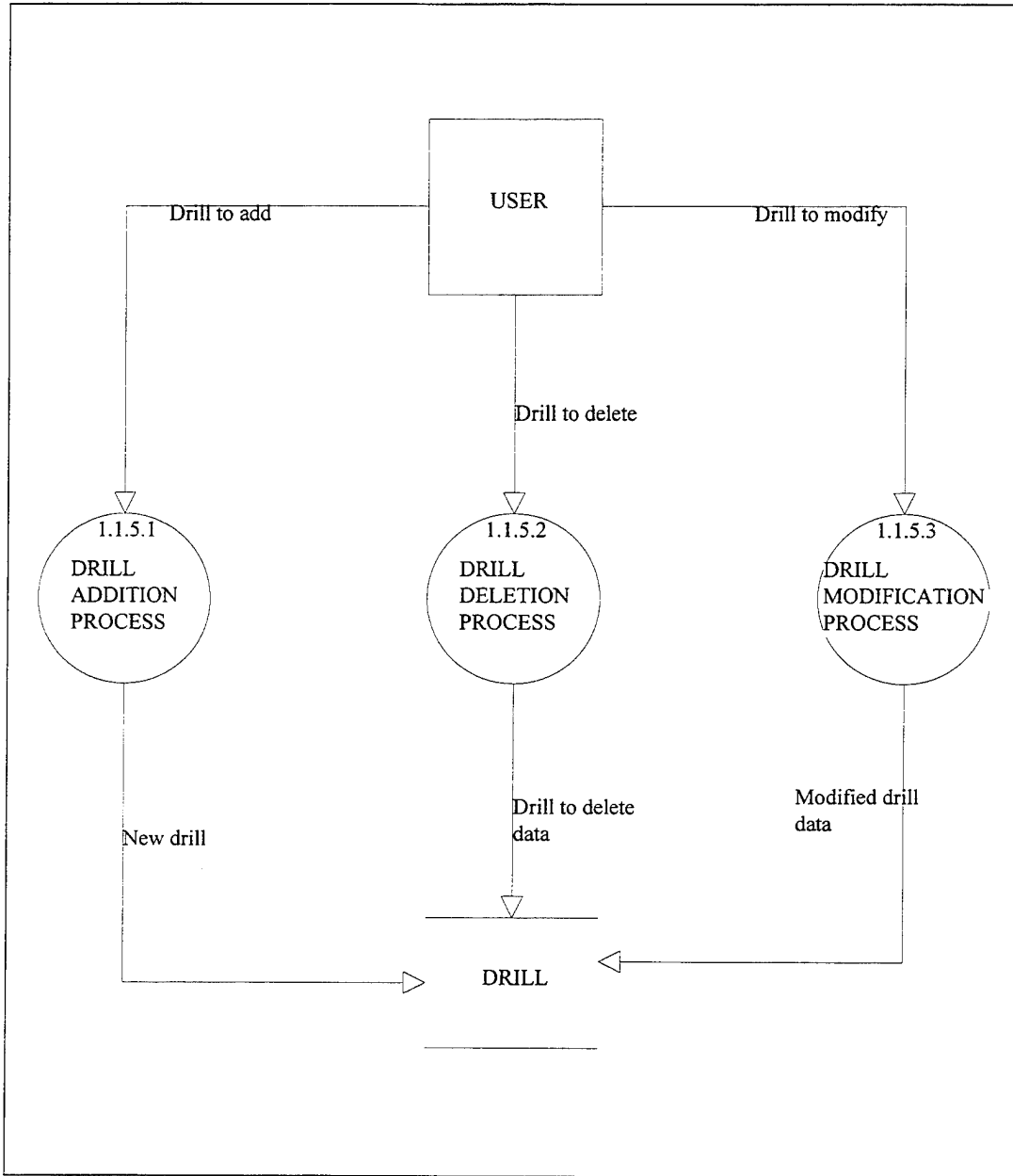




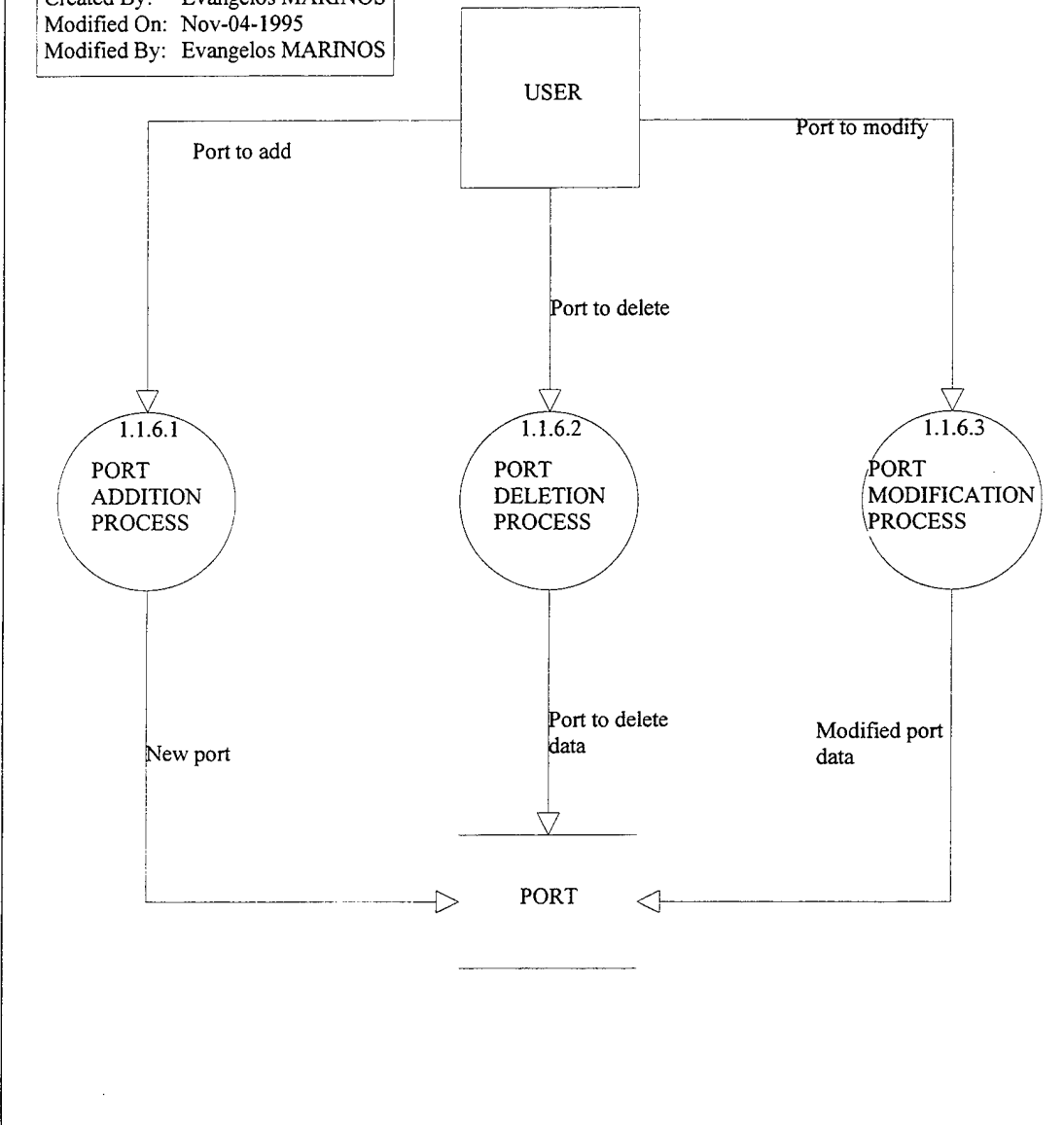


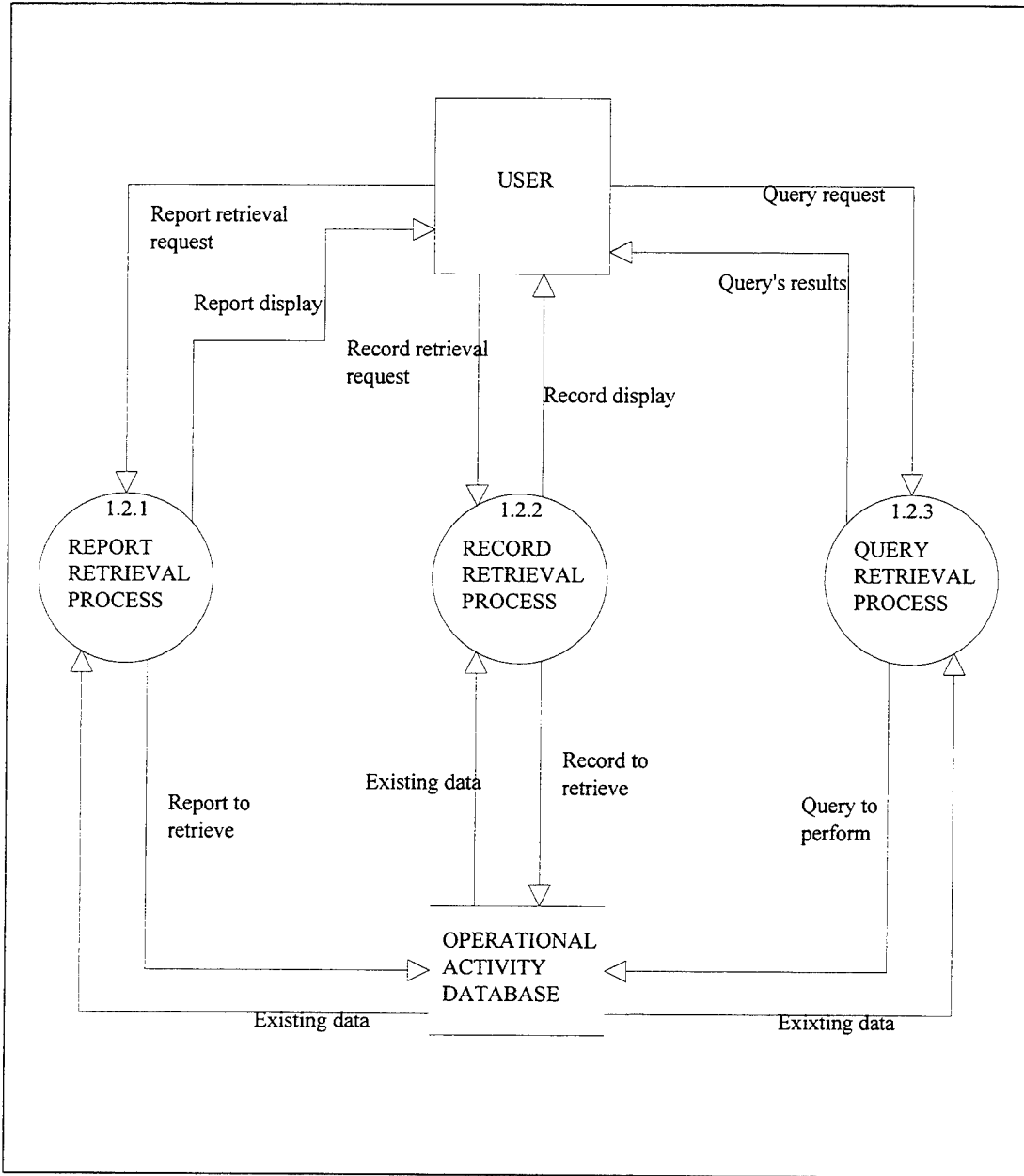


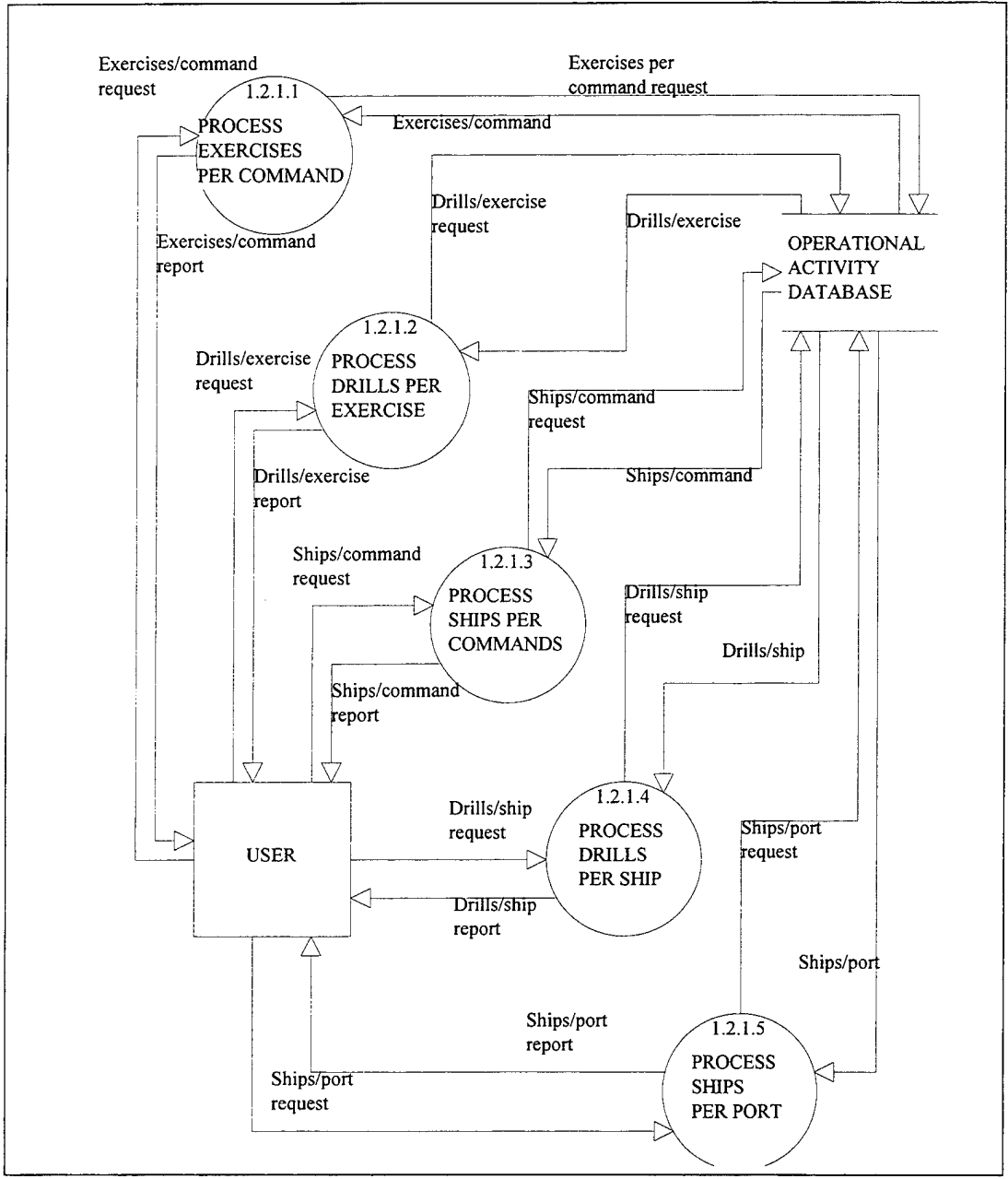


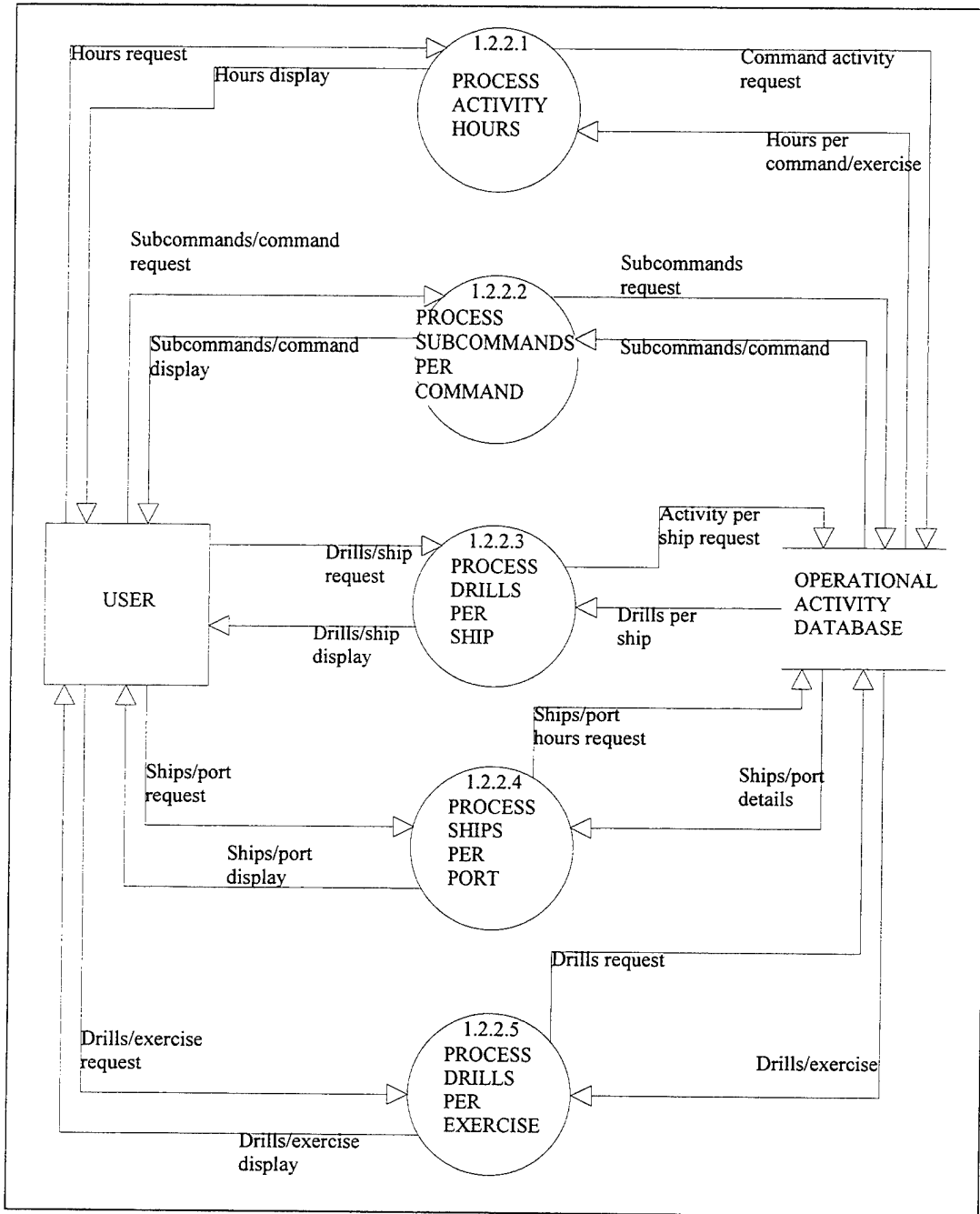


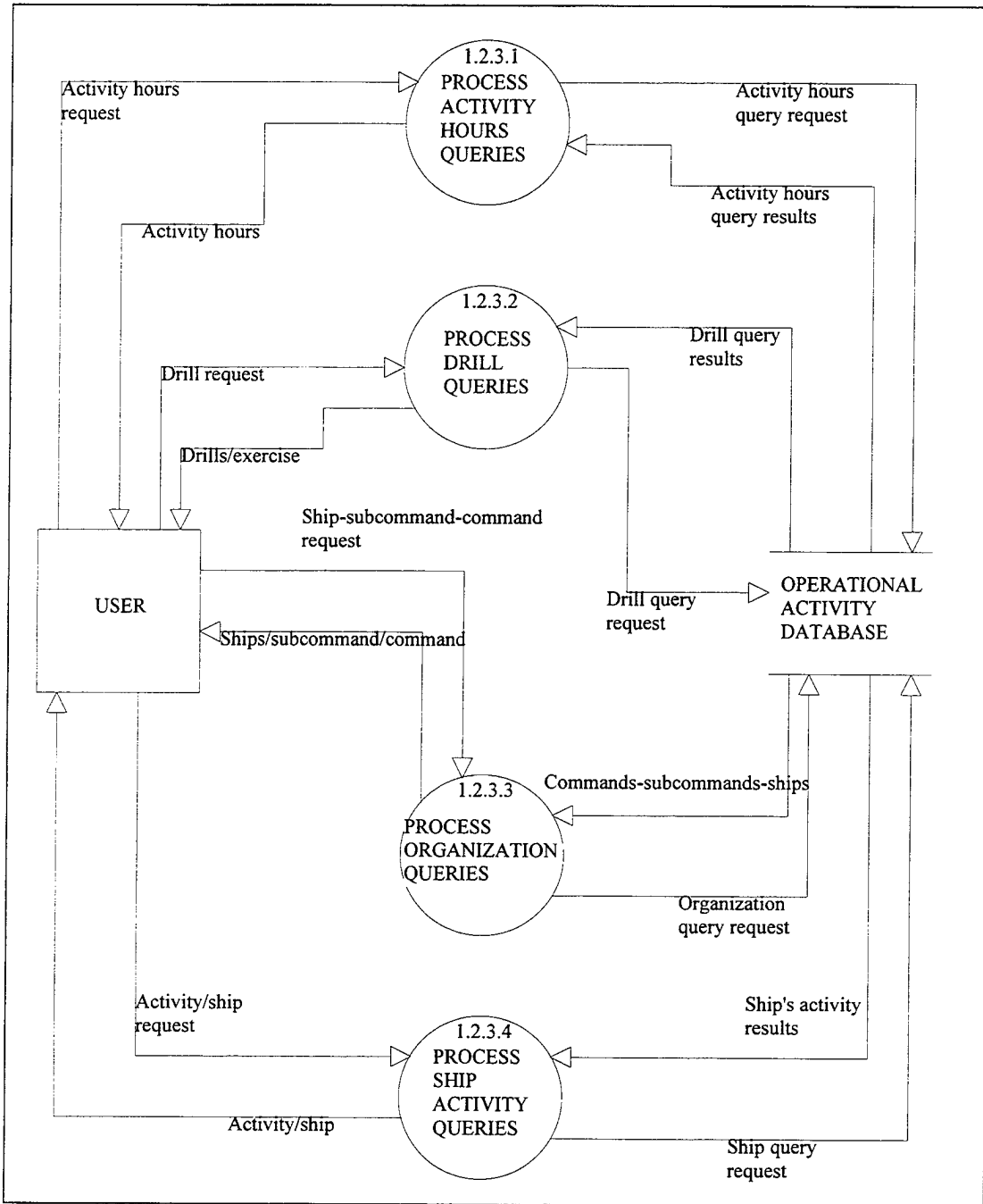
Project Name: Activities
Project Path: c:\ecwin\
Chart File: dfd00007.dfd
Chart Name: UPDATE PORT
Created On: Nov-04-1995
Created By: Evangelos MARINOS
Modified On: Nov-04-1995
Modified By: Evangelos MARINOS

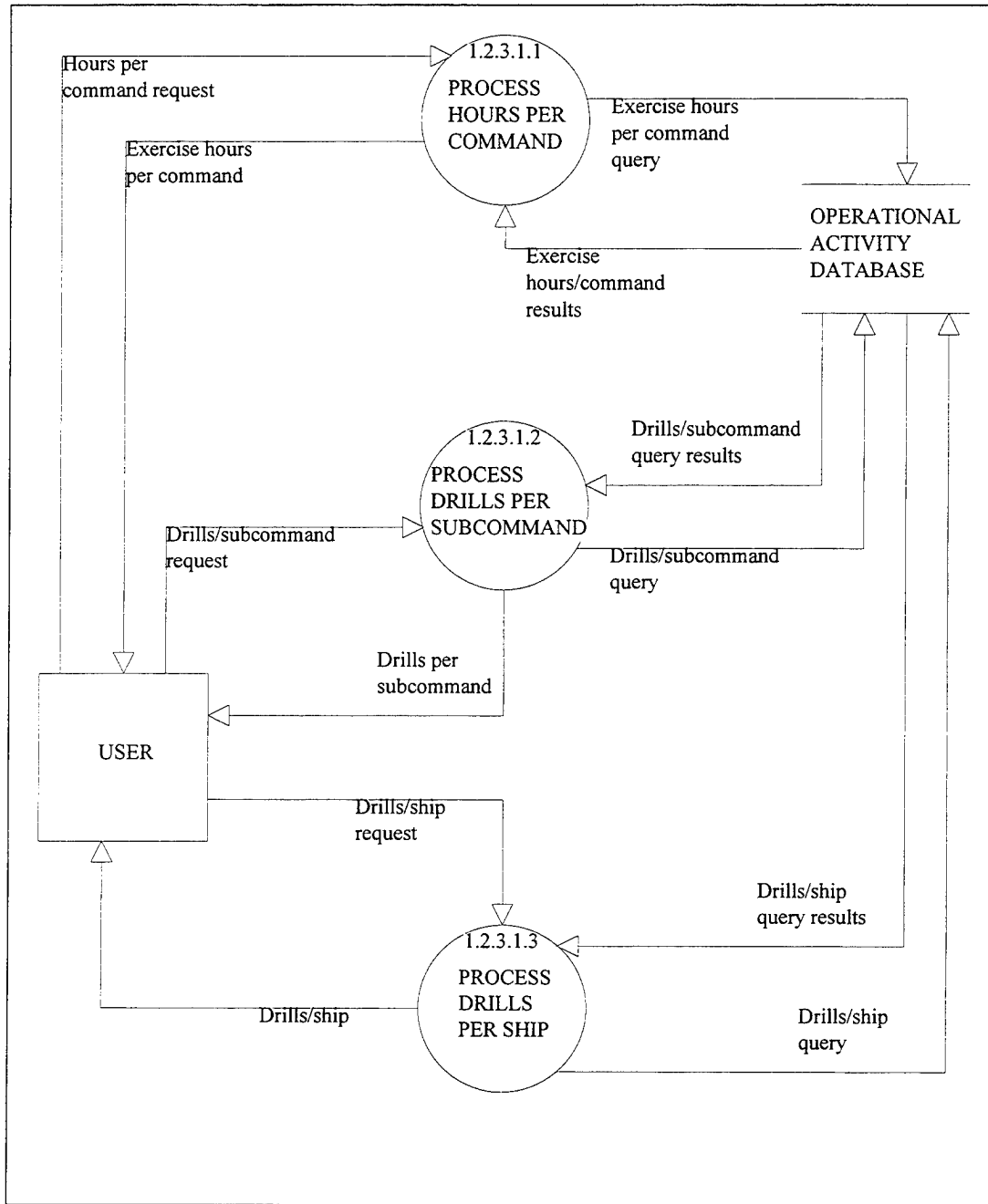


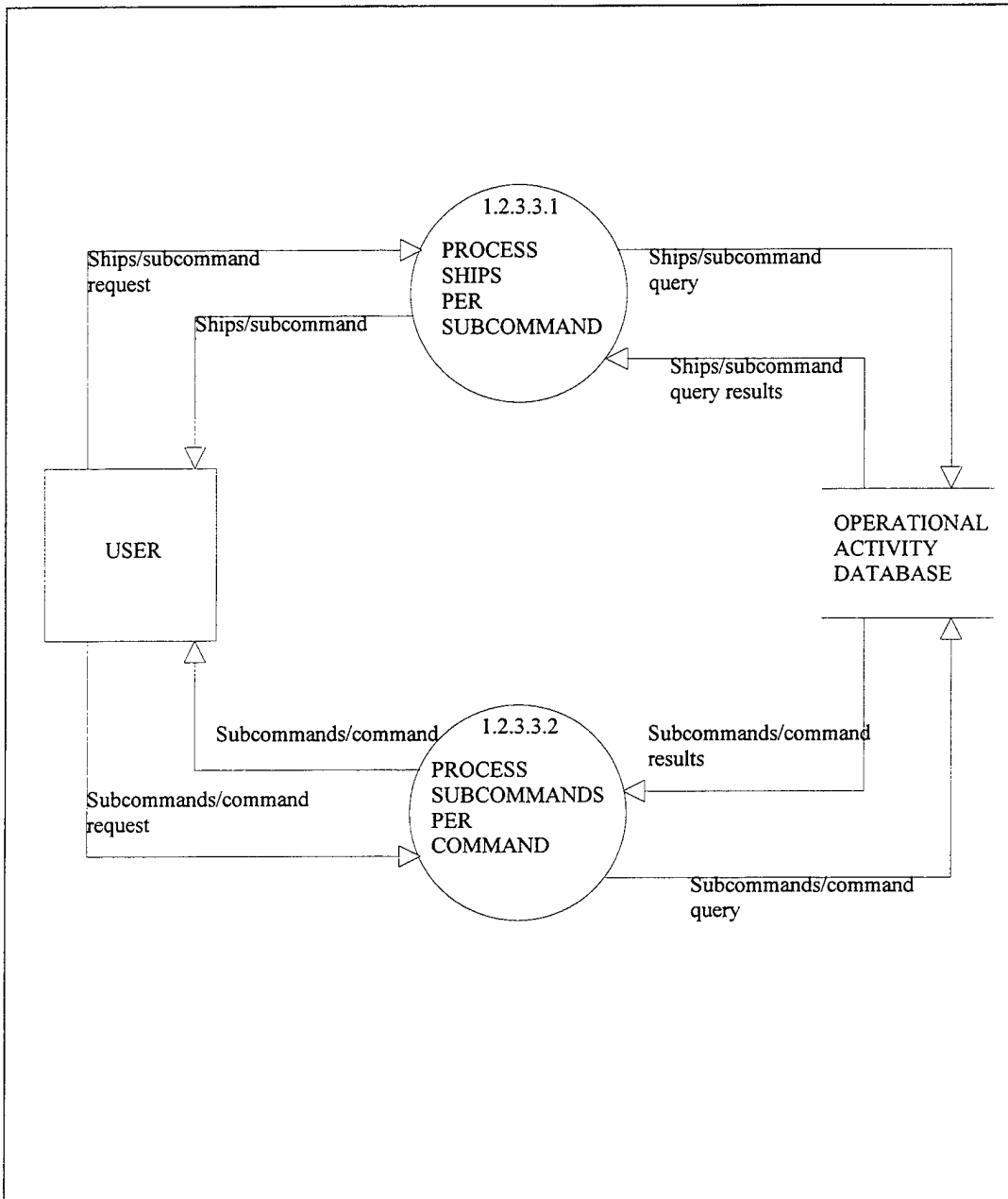












APPENDIX C : RELATIONS AND RELATIONAL MODEL

RELATIONS

COMMMAND (Command Name¹ , Commander Name, Commander Rank, Base Location)

SUBCOMMAND (Subcommand Name, *Command name*² , Subcommander Name, Subcommander Rank, Base Location)

EXERCISE (Exercise No, Exercise Name, Date begins, Date ends, Geogr. Location)

DRILL (Drill No, *Exercise No*, Drill Type, Drill Date, Time begins, Time ends, Objective)

SHIP (Hull No, *Subcommand Name*, Name, Type, CO_Name, CO_Rank, Number of personnel)

PORT (Port No, Port Name, Location, Watering Capab., Fueling Capab., Maintenance Capab.)

SHIP-PORT (Hull No, Port No, Date begins, Hours, Amount of Fuel in lt, Ship's state)

COMMAND-EXERCISE (Exercise No, Command Name, Hours)

SUBCOMMAND-DRILL (Subcommand Name, Drill No, Hours)

SHIP-DRILL (Hull No, Drill No, Hours, Torpedos used, A/A Missiles used, A/S Missiles used, Submarines detected).

¹ All the underlined Attributes are the key Attributes for the Relations

² All the Italic Attributes are the foreign key Attributes for the Relations

RELATIONAL MODEL

COMMAND

<u>Command Name</u>	Commander Name	Commander Rank	Base Location
---------------------	----------------	----------------	---------------

COMMAND-EXERCISE

<u>Exercise No</u>	<u>Command Name</u>	Hours
--------------------	---------------------	-------

SUBCOMMAND

<u>Subcommand Name</u>	<u>Command Name</u>	Subcommander Name
------------------------	---------------------	-------------------

SHIP

<u>Hull No</u>	<u>Subcommand Name</u>	Type	CO_Name
----------------	------------------------	------	---------

SUBCOMMAND-DRILL

<u>Subcommand Name</u>	<u>Drill No</u>	Hours
------------------------	-----------------	-------

EXERCISE

<u>Exercise No</u>	Exercise Name	Date begins	Date ends	Geogr. Location
--------------------	---------------	-------------	-----------	-----------------

DRILL

<u>Drill No</u>	<u>Exercise No</u>	Drill Type	Time begins	Objective
-----------------	--------------------	------------	-------------	-----------

SHIP-DRILL

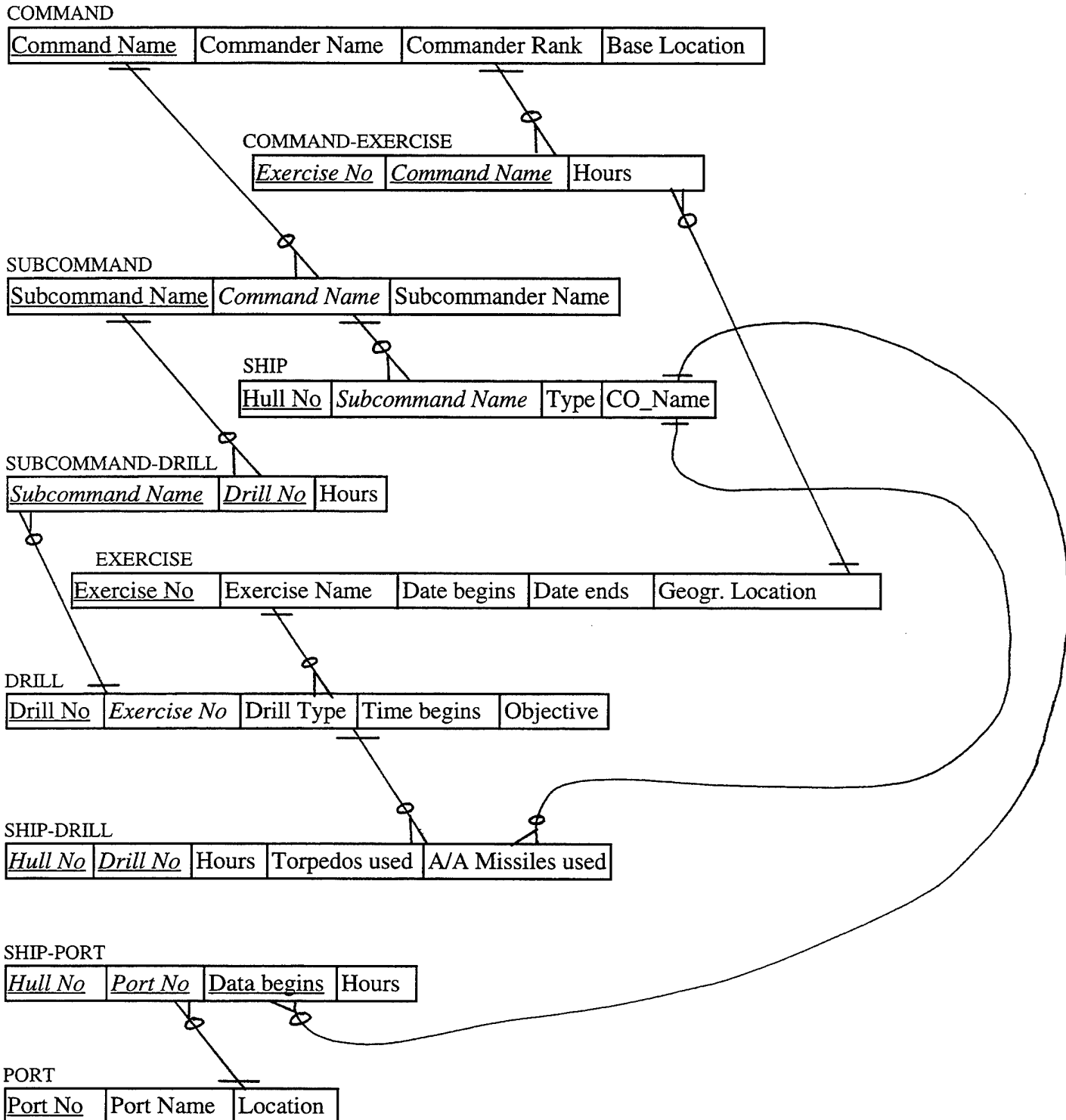
<u>Hull No</u>	<u>Drill No</u>	Hours	Torpedos used	A/A Missiles used
----------------	-----------------	-------	---------------	-------------------

SHIP-PORT

<u>Hull No</u>	<u>Port No</u>	<u>Data begins</u>	Hours
----------------	----------------	--------------------	-------

PORT

<u>Port No</u>	Port Name	Location
----------------	-----------	----------



APPENDIX D : BUSINESS RULES

ENTITY COMMAND

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Command Name	Must be unique	Non-null	Command name
Commander Name	Nonunique	May be null	Name
Commander Rank	Nonunique	May be null	Rank
Base Location	Nonunique	May be null	Location

ENTITY SUBCOMMAND

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Subcommand Name	Must be unique	Non-null	Subcommand name
Command Name	Nonunique	Non-null	Command name
Subcommander Name	Nonunique	May be null	Name
Subcommander Rank	Nonunique	May be null	Rank
Base Location	Nonunique	May be null	Location

User rule: SUBCOMMAND.Subcommander Rank not exceed COMMAND.Commander Rank

Event: Insert

Condition: SUBCOMMAND.Subcommander Rank > COMMAND.Commander Rank

Where SUBCOMMAND.Command Name = COMMAND.Command Name

Action: Reject the insert transaction

User rule: SUBCOMMAND.Command Name exists in COMMAND.Command Name

Event: Insert

Condition: SUBCOMMAND.Command Name not in {COMMAND.Command Name}

Action: Reject the insert transaction

ENTITY EXERCISE

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Exercise No	Must be unique	Non-null	Exercise #
Exercise Name	Nonunique	May be null	Exercise name
Geogr. Location	Nonunique	May be null	Location
Date begins	Nonunique	May be null	Date
Date ends	Nonunique	May be null	Date

ENTITY DRILL

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Drill No	Must be unique	Non-null	Drill #
Exercise No	Nonunique	Non-null	Exercise #
Drill Type	Nonunique	May be null	Drill type
Drill Date	Nonunique	May be null	Date
Time begins	Nonunique	May be null	Time
Time ends	Nonunique	May be null	Time
Objective	Nonunique	May be null	Objective

User rule: DRILL.Drill Date not exceed EXERCISE.Date ends

Event: Insert

Condition: DRILL.Drill Date > EXERCISE.Date ends

Where DRILL.Exercise No = EXERCISE.Exercise No

Action: Reject the insert transaction

User rule: DRILL.Drill Date not precede EXERCISE.Date begins

Event: Insert

Condition: DRILL.Drill Date < EXERCISE.Date begins

Where DRILL.Exercise No = EXERCISE.Exercise No

Action: Reject the insert transaction

User rule: DRILL.Exercise No exists in EXERCISE.Exercise No

Event: Insert

Condition: DRILL.Exercise No not in {EXERCISE.Exercise No}

Action: Reject the insert transaction

ENTITY SHIP

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Hull No	Must be unique	Non-null	Hull #
Subcommand Name	Nonunique	Non-null	Subcommand Name
Name	Nonunique	May be null	Ship Name
Type	Nonunique	May be null	Ship Type
CO_Name	Nonunique	May be null	Name
CO_Rank	Nonunique	May be null	Rank
Number of personnel	Nonunique	May be null	Number

User rule: SHIP.CO_Rank not exceed SUBCOMMAND.Subcommander Rank

Event: Insert

Condition: Ship.CO_Rank > SUBCOMMAND.Subcommander Rank

Where SHIP.Subcommand Name = SUBCOMMAND.Subcommand Name

Action: Reject the insert transaction

User rule: SHIP.Subcommand Name exists in SUBCOMM.Subcommand Name

Event: Insert

Condition: SHIP.Subcommand Name not in {SUBCOMMAND.Subcommand Name}

Action: Reject the insert transaction

ENTITY PORT

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Port No	Must be unique	Non-null	Port #
Port Name	Nonunique	May be null	Port name
Location	Nonunique	May be null	Location
Watering Capab.	Nonunique	May be null	Logical{Y,N}
Fueling Capab.	Nonunique	May be null	Logical{Y,N}
Maintenance capab.	Nonunique	May be null	Logical{Y,N}

ENTITY SHIP-PORT

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Port No	Nonunique	Non-null	Port #
Hull No	Nonunique	Non-null	Hull #
Date begins	Nonunique	Non-null	Date
Hours	Nonunique	May be null	Number
Amount of Fuel in lt	Nonunique	May be null	Number
Ship's state	Nonunique	May be null	Ship state

User rule: SHIP-PORT.Hull No exists in SHIP.Hull No

Event: Insert

Condition: SHIP-PORT.Hull No not in {SHIP.Hull No}

Action: Reject the insert transaction

User rule: SHIP-PORT.Drill No exists in DRILL.Drill No

Event: Insert

Condition: SHIP-PORT.Drill No not in {DRILL.Drill No}

Action: Reject the insert transaction

ENTITY COMMAND-EXERCISE

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Exercise No	Nonunique	Non-null	Exercise #
Command Name	Nonunique	Non-null	Command name
Hours	Nonunique	May be null	Number

User rule: COMMAND-EXERCISE.Command Name exists in COMMAND.Command Name

Event: Insert

Condition: COMMAND-EXERCISE.Command Name not in {COMMAND.Command Name}

Action: Reject the insert transaction

User rule: COMMAND-EXERCISE.Exercise No exists in EXERCISE.Exercise No

Event: Insert

Condition: COMMAND-EXERCISE.Exercise No not in {EXERCISE.Exercise No}

Action: Reject the insert transaction

ENTITY SUBCOMMAND-DRILL

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Drill No	Nonunique	Non-null	Drill #
Subcommand Name	Nonunique	Non-null	Subcommand name
Hours	Nonunique	May be null	Number

User rule: SUBCOMMAND-DRILL.Subcommand Name exists in SUBCOMMAND.Subcommand Name

Event: Insert

Condition: SUBCOMMAND-DRILL.Subcommand Name not in
{SUBCOMMAND.Subcommand Name}

Action: Reject the insert transaction

User rule: SUBCOMMAND-DRILL.Drill No exists in DRILL.Drill No

Event: Insert

Condition: SUBCOMMAND-DRILL.Drill No not in {DRILL.Drill No}

Action: Reject the insert transaction

ENTITY SHIP-DRILL

	<u>Uniqueness</u>	<u>Null Support</u>	<u>Domain</u>
Hull No	Nonunique	Non-null	Hull #
Drill No	Nonunique	Non-null	Drill #
Hours	Nonunique	May be null	Number
Torpedos used	Nonunique	May be null	Number
A/A Missiles used	Nonunique	May be null	Number
A/S Missiles used	Nonunique	May be null	Number
Submarines detected	Nonunique	May be null	Number

User rule: SHIP-DRILL.Drill No exists in DRILL.Drill No

Event: Insert

Condition: SHIP-DRILL.Drill No not in {DRILL.Drill No}

Action: Reject the insert transaction

User rule: SHIP-DRILL.Hull No exists in SHIP.Hull No

Event: Insert

Condition: SHIP-DRILL.Hull No not in {SHIP.Hull No}

Action: Reject the insert transaction

User rule: SHIP-DRILL.Hull No not same with SHIP-PORT.Hull No

Event: Insert

Condition: SHIP-DRILL.Hull No = SHIP-PORT.Hull No

Where DRILL.Drill Date = SHIP-PORT.Date begins

and DRILL.Drill No = SHIP-DRILL.Drill No

Action: Reject the insert transaction

APPENDIX E : PROCEDURES FOR INSTALLING AND OPERATING OADS

A. INSTALLATION PROCEDURE

To run any Paradox for Windows 5.0 application, Paradox 5.0 itself must be installed. To install Paradox for Windows 5.0 the user has to run the Paradox 5.0 installation program. Instructions for installing Paradox 5.0 can be found in Paradox 5.0 manuals.

B. SETTING UP THE OADS APPLICATION

After installing Paradox for Windows 5.0 you need to setup the OADS application. Your OADS application disk includes all the required files for setting up the OADS application. The user has to perform the following steps:

- Make sure that you are in the Windows environment.
- In the Windows click the *File Manager* icon
- In the File Manager environment, insert the OADS application disk, and copy the files of the disk to *C:\pdowin\working* directory
- Exit the File Manager, and click the *Paradox for Windows* icon to enter the Paradox for Windows environment
- In the Paradox for Windows environment, go to the *working* directory

- In the working directory, open the *Setup.fsl* file, in order to go to the OADS main menu screen

C. USING THE OADS APPLICATION

Using the mouse, you can choose the Update Submenu, or the Retrieval Submenu.

If you want to update (add, delete, or modify) data of the OADS application, you have to click the Update Subsystem button, which bring you to a list of buttons (Command, Subcommand, Ship, Exercise, Drill, etc.). By pressing the appropriate button of the list, you are transferred to the data that you want to update.

If you want to retrieve data from the OADS application, you have to click the Retrieval Subsystem button, which brings you to a selection list of Reports, Records, and Queries. By pressing the appropriate button, you are transferred to the data that you want to retrieve.

1. Help Screens

Help screens are included in the OADS application and are designed to help the user follow the right steps for each procedures. In this way the user can respond correctly to data entry and updates and eliminate potential mistakes.

2. Printing Outputs

After performing a report operation from the Reports selection of the Retrieval Subsystem, there is the opportunity to print the results to the printer.

APPENDIX F : APPLICATION CODE

Main Menu

```
method pushButton(var eventInfo Event)
var custForm Form endVar
custForm.open("F18")
formReturn("OK")
endmethod
```

```
method pushButton(var eventInfo Event)
var custForm Form endVar
custForm.open("F7")
formReturn("OK")
endmethod
```

Update Subsystem

```
method pushButton(var eventInfo Event)
var custForm Form endVar
custForm.open("F8")
endmethod
```

```
method pushButton(var eventInfo Event)
var custForm Form endVar
custForm.open("F11")
endmethod
```

```
method pushButton(var eventInfo Event)
var custForm Form endVar
```

```
custForm.open("F10")  
endmethod
```

```
method pushButton(var eventInfo Event)  
var custForm Form endVar  
custForm.open("F12")  
endmethod
```

```
method pushButton(var eventInfo Event)  
var custForm Form endVar  
custForm.open("F13")  
endmethod
```

```
method pushButton(var eventInfo Event)  
var custForm Form endVar  
custForm.open("F16")  
endmethod
```

```
method pushButton(var eventInfo Event)  
var custForm Form endVar  
custForm.open("F14")  
endmethod
```

```
method pushButton(var eventInfo Event)  
var custForm Form endVar  
custForm.open("F15")  
endmethod
```

```
method pushButton(var eventInfo Event)
var custForm Form endVar
custForm.open("F9")
endmethod
```

```
method pushButton(var eventInfo Event)
var custForm Form endVar
custForm.open("F17")
endmethod
```

```
method pushButton(var eventInfo Event)
var custForm Form endVar
custForm.open("startup")
close()
formReturn("OK")
endmethod
```

Command Update

```
method arrive(var eventInfo MoveEvent)
var
    mainMenu Menu
    filePop, editPop, recordPop PopUpMenu
endvar
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
```

```
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod
```

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endif
endif
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
action(DataCancelRecord)
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
var
```

```
userInput, promptString String
```

```
endVar
```

```
promptString = "Enter the Command Name here."
```

```
userInput = promptString
```

```
userInput.view("What is the Command Name?")
```

```
if userInput <> promptString then
```

```
  if not Command_Name.locate("Command Name", userInput) then
```

```
    beep( )
```

```
    message("Couldn't find", userInput)
```

```
    sleep(1000)
```

```
  endIf
```

```
endIf
```

```
endmethod
```

Subcommand Update

```
method arrive(var eventInfo MoveEvent)
```

```
var
```

```
  mainMenu Menu
```

```
  filePop, editPop, recordPop PopUpMenu
```

```
endvar
```



```
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod
```

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method open(var eventInfo Event)
```

```
doDefault
self.DataSource = "[COMMAND.Command Name]"
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataCancelRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
var
userInput, promptString String
endVar
promptString = "Enter the Subcommand Name here."
userInput = promptString
userInput.view("What is the Subcommand Name?")
if userInput <> promptString then
  if not Subcommand_Name.locate("Subcommand Name", userInput) then
    beep( )
    message("Couldn't find", userInput)
    sleep(1000)
```

```
    endIf
endIf
endmethod

method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Ship Update

```
method arrive(var eventInfo MoveEvent)
var
    mainMenu Menu
    filePop, editPop, recordPop PopUpMenu
endvar
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
```

```
mainMenu.show ( )  
endmethod
```

```
method action(var eventInfo ActionEvent)  
if eventInfo.id( ) = DataDeleteRecord then  
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then  
doDefault  
else  
eventInfo.setErrorCode(UserError)  
endif  
endif  
endmethod
```

```
method pushButton(var eventInfo Event)  
action(DataBeginEdit)  
endmethod
```

```
method pushButton(var eventInfo Event)  
action(DataEndEdit)  
endmethod
```

```
method pushButton(var eventInfo Event)  
action(DataCancelRecord)  
endmethod
```

```
method pushButton(var eventInfo Event)  
var  
userInput, promptString String
```

```
endVar
promptString = "Enter the Hull Number here."
userInput = promptString
userInput.view("What is the Hull Number?")
if userInput <> promptString then
  if not Hull_No.locate("Hull No", userInput) then
    beep( )
    message("Couldn't find", userInput)
    sleep(1000)
  endIf
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Port Update

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method arrive(var eventInfo MoveEvent)
var
    mainMenu Menu
    filePop, editPop, recordPop PopUpMenu
endvar
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
action(DataCancelRecord)
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
var
```

```
userInput, promptString String
```

```
endVar
```

```
promptString = "Enter the Port Number here."
```

```
userInput = promptString
```

```
userInput.view("What is the Port Number?")
```

```
if userInput <> promptString then
```

```
  if not Port_No.locate("Port No", userInput) then
```

```
    beep( )
```

```
    message("Couldn't find", userInput)
```

```
    sleep(1000)
```

```
  endif
```

```
endif
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
formReturn("OK")
```

```
endmethod
```

Ship Per Port Update

```
method action(var eventInfo ActionEvent)
```

```
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method changeValue(var eventInfo ValueEvent)
if eventInfo.newValue( ) > Today( ) then
eventInfo.setErrorCode(CanNotDepart)
message("Date can't be later than today's date.")
sleep(1000)
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
```



```
action(DataBeginEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataCancelRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
var
userInput1, promptString1 String
userInput2, promptString2 String
endVar
promptString1 = "Enter the Hull Number here."
userInput1 = promptString1
promptString2 = "Enter the Port Number here."
userInput2 = promptString2
userInput1.view("What is the Hull Number?")
userInput2.view("What is the Port Number?")
if userInput1 <> promptString1 and userInput2 <> promptString2 then
  if not Hull_No.locate("Hull No", userInput1) then
    beep()
    message("Couldn't find", userInput1)
    else if not Port_No.locate("Port No", userInput2) then
      beep()
```

```
        message("Couldn't find", userInput2)
    endif
endIf
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Exercise Per Command Update

```
method action(var eventInfo ActionEvent)
if eventInfo.id() = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method arrive(var eventInfo MoveEvent)
var
    mainMenu Menu
    filePop, editPop, recordPop PopUpMenu
endvar
filePop.addText ("New")
filePop.addText ("Exit")
```

```
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod
```

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endif
endif
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataCancelRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
var
userInput1, promptString1 String
userInput2, promptString2 String
endVar
promptString1 = "Enter the Command Name here."
userInput1 = promptString1
promptString2 = "Enter the Exercise Number here."
userInput2 = promptString2
userInput1.view("What is the Command Name?")
userInput2.view("What is the Exercise Number?")
if userInput1 <> promptString1 and userInput2 <> promptString2 then
  if not Command_Name.locate("Command Name", userInput1) then
    beep()
    message("Couldn't find", userInput1)
    else if not Exercise_No.locate("Exercise No", userInput2) then
      beep()
      message("Couldn't find", userInput2)
      sleep(1000)
```

```
        endIf
    endIf
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Drill Per Subcommand Update

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
```

```

endmethod

method arrive(var eventInfo MoveEvent)
var
    mainMenu Menu
    filePop, editPop, recordPop PopUpMenu
endvar
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod

method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod

```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataCancelRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
var
userInput1, promptString1 String
userInput2, promptString2 String
endVar
promptString1 = "Enter the Subcommand Name here."
userInput1 = promptString1
promptString2 = "Enter the Drill Number here."
userInput2 = promptString2
userInput1.view("What is the Subcommand Name?")
userInput2.view("What is the Drill Number?")
if userInput1 <> promptString1 and userInput2 <> promptString2 then
  if not Subcommand_Name.locate("Subcommand Name", userInput1) then
    beep( )
    message("Couldn't find", userInput1)
    else if not Drill_No.locate("Drill No", userInput2) then
      beep( )
      message("Couldn't find", userInput2)
      sleep(1000)
    endif
  endif
endIf
```

```
    endIf
endIf
endmethod

method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Drill Per Ship Update

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```



```

method pushButton(var eventInfo Event)
var
  userInput1, promptString1 String
  userInput2, promptString2 String
endVar
promptString1 = "Enter the Hull Number here."
userInput1 = promptString1
promptString2 = "Enter the Drill Number here."
userInput2 = promptString2
userInput1.view("What is the Hull Number?")
userInput2.view("What is the Drill Number?")
if userInput1 <> promptString1 and userInput2 <> promptString2 then
  if not Hull_No.locate("Hull No", userInput1) then
    beep()
    message("Couldn't find", userInput1)
    else if not Drill_No.locate("Drill No", userInput2) then
      beep()
      message("Couldn't find", userInput2)
      sleep(1000)
    endIf
  endIf
endIf
endmethod

method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod

```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataCancelRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Exercise Update

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endif
endif
endmethod
```

```
method arrive(var eventInfo MoveEvent)
var
    mainMenu Menu
    filePop, editPop, recordPop PopUpMenu
```

```
endvar
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod
```

```
method changeValue(var eventInfo ValueEvent)
if eventInfo.newValue( ) > Today( ) then
eventInfo.setErrorCode(CanNotDepart)
message("Date can't be later than today's date.")
sleep(1000)
endif
endmethod
```

```
method changeValue(var eventInfo ValueEvent)
if eventInfo.newValue( ) > Today( ) then
```

```
eventInfo.setErrorCode(CanNotDepart)
message("Date can't be later than today's date.")
sleep(1000)
endIf
if eventInfo.newValue( ) < "[EXERCISE.Date begins]" then
eventInfo.setErrorCode(CanNotDepart)
message("Date can't be earlier than Date begins.")
sleep(1000)
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataCancelRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
var
userInput, promptString String
endVar
promptString = "Enter the Exercise Number here."
```

```

userInput = promptString
userInput.view("What is the Exercise Number?")
if userInput <> promptString then
  if not Exercise_No.locate("Exercise No", userInput) then
    beep( )
    message("Couldn't find", userInput)
    sleep(1000)
  endif
endif
endmethod

```

```

method arrive(var eventInfo MoveEvent)
var
  mainMenu Menu
  filePop, editPop, recordPop PopUpMenu
endvar
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")

```

```
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Drill Update

```
method action(var eventInfo ActionEvent)
if eventInfo.id( ) = DataDeleteRecord then
if msgQuestion ("Delete?", "Delete this record?") = "Yes" then
doDefault
else
eventInfo.setErrorCode(UserError)
endIf
endIf
endmethod
```

```
method changeValue(var eventInfo ValueEvent)
if eventInfo.newValue( ) > Today( ) then
eventInfo.setErrorCode(CanNotDepart)
message("Date can't be later than today's date.")
sleep(1000)
endIf
if eventInfo.newValue( ) < "[EXERCISE.Date begins]" then
eventInfo.setErrorCode(CanNotDepart)
```

```
message("Date can't be earlier than exercise's date.")
sleep(1000)
endIf
if eventInfo.newValue( ) > "[EXERCISE.Date ends]" then
eventInfo.setErrorCode(CanNotDepart)
message("Date can't be later than exercise's date.")
sleep(1000)
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataBeginEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataEndEdit)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataCancelRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
var
userInput, promptString String
endVar
promptString = "Enter the Drill Number here."
userInput = promptString
```

```
userInput.view("What is the Drill Number?")
if userInput <> promptString then
  if not Drill_No.locate("Drill No", userInput) then
    beep( )
    message("Couldn't find", userInput)
    sleep(1000)
  endIf
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

```
method arrive(var eventInfo MoveEvent)
var
  mainMenu Menu
  filePop, editPop, recordPop PopUpMenu
endvar
filePop.addText ("New")
filePop.addText ("Exit")
mainMenu.addPopUp("File", filePop)
editPop.addText ("Cut")
editPop.addText ("Copy")
editPop.addText ("Paste")
mainMenu.addPopUp("Edit", editPop)
recordPop.addText ("Next")
recordPop.addText ("Prev")
```



```
recordPop.addSeparator ( )
recordPop.addText ("Edit Data")
recordPop.addText ("Insert")
recordPop.addText ("Delete")
mainMenu.addPopUp("Record", recordPop)
mainMenu.show ( )
endmethod
```

Retrieval Subsystem

```
method pushButton(var eventInfo Event)
var
    custForm Form
endVar
custForm.open("F19")
endmethod
```

```
method pushButton(var eventInfo Event)
var
    custForm Form
endVar
custForm.open("F20")
endmethod
```

```
method pushButton(var eventInfo Event)
var
    custForm Form
endVar
custForm.open("F21")
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
var
```

```
    custForm Form
```

```
endVar
```

```
custForm.open("startup")
```

```
close()
```

```
formReturn("OK")
```

```
endmethod
```

Reports

```
method pushButton(var eventInfo Event)
```

```
var
```

```
    custRpt Report
```

```
endVar
```

```
custRpt.open("r1")
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
var
```

```
    custRpt Report
```

```
endVar
```

```
custRpt.print("r1")
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
var
```

```
        custRpt Report
    endVar
    custRpt.open("R2")
endmethod

method pushButton(var eventInfo Event)
var
    custRpt Report
endVar
    custRpt.print("r2")
endmethod

method pushButton(var eventInfo Event)
var
    custRpt Report
endVar
    custRpt.open("R3")
endmethod

method pushButton(var eventInfo Event)
var
    custRpt Report
endVar
    custRpt.print("r3")
endmethod

method pushButton(var eventInfo Event)
var
```

```
        custRpt Report
endVar
custRpt.open("r4")
endmethod

method pushButton(var eventInfo Event)
var
        custRpt Report
endVar
custRpt.print("r4")
endmethod

method pushButton(var eventInfo Event)
var
        custRpt Report
endVar
custRpt.open("r5")
endmethod

method pushButton(var eventInfo Event)
var
        custRpt Report
endVar
custRpt.print("r5")
endmethod

method pushButton(var eventInfo Event)
var
```

```
        custForm Form
endVar
custForm.open("F7")
formReturn("OK")
endmethod
```

Records

```
method pushButton(var eventInfo Event)
var
        custForm Form
endVar
custForm.open("F1")
endmethod
```

```
method pushButton(var eventInfo Event)
var
        custForm Form
endVar
custForm.open("F3")
endmethod
```

```
method pushButton(var eventInfo Event)
var
        custForm Form
endVar
custForm.open("F6")
endmethod
```

```
method pushButton(var eventInfo Event)
var
    custForm Form
endVar
custForm.open("F2")
endmethod
```

```
method pushButton(var eventInfo Event)
var
    custForm Form
endVar
custForm.open("F5")
endmethod
```

```
method pushButton(var eventInfo Event)
var
    custForm Form
endVar
custForm.open("F4")
endmethod
```

```
method pushButton(var eventInfo Event)
var
    custForm Form
endVar
custForm.open("F7")
formReturn("OK")
endmethod
```

Queries

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q12.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endif
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q13.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endif
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
```

```
if ExecuteQBFile("q2.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q7.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q3.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```



```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q8.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q10.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q9.qbe", "Answer.db") then
```

```
        tv.open("Answer.db")
    else
        msgStop("Stop", "Couldn't run the query.")
    endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q14.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q1.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q11.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q5.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q4.qbe", "Answer.db") then
    tv.open("Answer.db")
```

```
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q15.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
var
tv TableView
endVar
if ExecuteQBFile("q6.qbe", "Answer.db") then
    tv.open("Answer.db")
else
    msgStop("Stop", "Couldn't run the query.")
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
var
    custForm Form
endVar
custForm.open("F7")
formReturn("OK")
endmethod
```

Command Records Per Exercise

```
method pushButton(var eventInfo Event)
action(DataNextRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataPriorRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
var
    userInput, promptString String
endVar
promptString = "Enter the Command Name here."
userInput = promptString
userInput.view("What is the Command Name?")
if userInput <> promptString then
    if not Command_Name.locate("Command Name", userInput) then
        beep( )
        message("Couldn't find", userInput)
        sleep(1000)
    endif
endif
endmethod
```

```
endif
endif
endmethod
```

Subcommand Records Per Command

```
method pushButton(var eventInfo Event)
var
userInput, promptString String
endVar
promptString = "Enter the Command Name here."
userInput = promptString
userInput.view("What is the Command Name?")
if userInput <> promptString then
  if not Command_Name.locate("Command Name", userInput) then
    beep()
    message("Couldn't find", userInput)
    sleep(1000)
  endif
endif
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataNextRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataPriorRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Drill Records Per Ship

```
method pushButton(var eventInfo Event)
var
userInput, promptString String
endVar
promptString = "Enter the Hull Number here."
userInput = promptString
userInput.view("What is the Hull Number?")
if userInput <> promptString then
  if not Hull_No.locate("Hull No", userInput) then
    beep( )
    message("Couldn't find", userInput)
    sleep(1000)
  endIf
endIf
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataNextRecord)
```

```
endmethod
```

```
method pushButton(var eventInfo Event)  
action(DataPriorRecord)  
endmethod
```

```
method pushButton(var eventInfo Event)  
formReturn("OK")  
endmethod
```

Ship Records Per Port

```
method pushButton(var eventInfo Event)  
var  
userInput, promptString String  
endVar  
promptString = "Enter the Port Number here."  
userInput = promptString  
userInput.view("What is the Port Number?")  
if userInput <> promptString then  
  if not Port_No.locate("Port No", userInput) then  
    beep( )  
    message("Couldn't find", userInput)  
    sleep(1000)  
  endIf  
endIf  
endmethod
```

```
method pushButton(var eventInfo Event)
```



```
formReturn("OK")
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
action(DataNextRecord)
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
```

```
action(DataPriorRecord)
```

```
endmethod
```

Drill Records Per Ship Per Subcommand

```
method pushButton(var eventInfo Event)
```

```
var
```

```
userInput, promptString String
```

```
endVar
```

```
promptString = "Enter the Subcommand Name here."
```

```
userInput = promptString
```

```
userInput.view("What is the Subcommand Name?")
```

```
if userInput <> promptString then
```

```
    if not Subcommand_Name.locate("Subcommand Name", userInput) then
```

```
        beep( )
```

```
        message("Couldn't find", userInput)
```

```
        sleep(1000)
```

```
    endIf
```

```
endIf
```

```
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataNextRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataPriorRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

Drill Records Per Exercise

```
method pushButton(var eventInfo Event)
var
userInput, promptString String
endVar
promptString = "Enter the Exercise Number here."
userInput = promptString
userInput.view("What is the Exercise Number?")
if userInput <> promptString then
  if not Exercise_No.locate("Exercise No", userInput) then
    beep( )
    message("Couldn't find", userInput)
    sleep(1000)
  endIf
endIf
endmethod
```

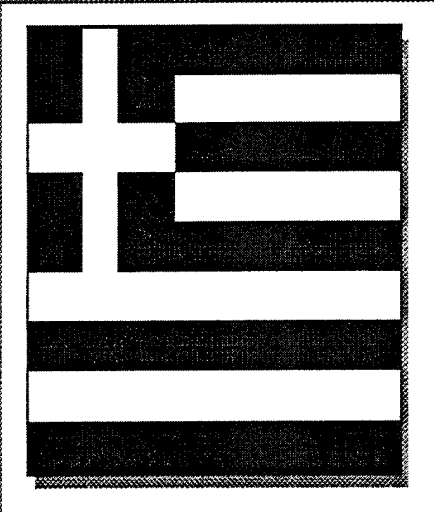
```
method pushButton(var eventInfo Event)
action(DataNextRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
action(DataPriorRecord)
endmethod
```

```
method pushButton(var eventInfo Event)
formReturn("OK")
endmethod
```

APPENDIX G: APPLICATION MENUS

Welcome to
**OPERATIONAL ACTIVITY
DATABASE SYSTEM**
by LT Evangelos P. Marinos



UPDATE SUBSYSTEM **RETRIEVAL SUBSYSTEM**

UPDATE SUBSYSTEM

COMMAND

SUBCOMMAND

SHIP

EXERCISE/COMMAND

EXERCISE/SUBCOMMAND

EXERCISE/SHIP

EXERCISE

SHIP

PORT

SHIP/PORT

GO BACK TO MAIN MENU

RETRIEVAL SUBSYSTEM

REPORTS

RECORDS

MISCELLANEOUS DATA

GO BACK TO MAIN MENU

Operational Activity System's REPORTS

COMMANDS AND EXERCISES PER COMMAND

Print

SHIPS PER SUBCOMMAND PER COMMAND

Print

DRILLS PER EXERCISE

Print

DRILLS PER SHIP

Print

SHIPS PER PORT

Print

GO BACK TO MAIN RETRIEVAL SUBMENU

Operational Activity System's RECORDS

HOURS PER COMMAND PER EXERCISE

DRILLS PER SHIP

DRILLS PER SHIP PER SUBCOMMAND

SUBCOMMANDS PER COMMAND

SHIPS PER PORT

DRILLS PER EXERCISE

GO BACK TO MAIN RETRIEVAL SUBMENU

Operational Activity System's QUERIES

EXERCISE HOURS PER COMMAND

DRILL HOURS PER SUBCOMMAND

HOURS PER DRILL PER SHIP

DRILLS PER EXERCISE

MAINTENANCE HOURS PER SHIP

FUEL TAKEN PER PORT PER SHIP

CAPABILITIES PER PORT

PORT HOURS PER SHIP

SUBMARINES DETECTED PER SHIP

WEAPONS USED PER SHIP PER DRILL

TORPEDOES USED PER SHIP PER DRILL

ANTISURFACE MISSILES USED PER SHIP

A/A MISSILES USED PER SHIP

SHIPS PER SUBCOMMAND

NUMBER OF SHIPS PER SUBCOMMAND PER COMMAND

GO BACK TO MAIN RETRIEVAL SUBMENU

LIST OF REFERENCES

1. Kroenke,D.M., *Database Processing Fundamentals, Design, and Implementation*, Prentice-Hall, Inc., 1995.
2. Tsongas, G. C., *Design and Implermntation of a Database System to Support Administrative Functions Aboard Hellenic Navy Vessels*, Naval Postgraduate School Thesis, 1994.
3. Whitten,J.L., Bentley, L. D., Barlow, V. M., *System Analysis and Design Methods*, Irwin, Inc., 1994.
4. Paradox for Windows v. 5.0, *User's Guide*, Borland International, Inc., 1994.

BIBLIOGRAPHY

1. Page-Jones, M., *The Practical Guide to Structured System Design*, Prentice-Hall, Inc., 1988.
2. Kroenke, D.M., *Database Processing Fundamentals, Design, and Implementation*, Prentice-Hall, Inc., 1995.
3. Mcfadden, F.R., Hoffer, J.A., *Modern Database Management*, Benjamin/Cummings, Inc., 1994.
4. Tsongas, G. C., *Design and Implementation of a Database System to Support Administrative Functions Aboard Hellenic Navy Vessels*, Naval Postgraduate School Thesis, 1994.
5. Paradox for Windows v. 5.0, *User's Guide*, Borland International, Inc., 1994.
6. Paradox for Windows v. 5.0, *Guide to Object PAL*, Borland International, Inc., 1994.
7. Paradox for Windows v. 5.0, *Workgroup Desktop Guide*, Borland International, Inc., 1994.
8. Paradox for Windows v. 5.0, *Object PAL Quick Reference*, Borland International, Inc., 1994.
9. Shneiderman, B., *Designing the User Interface*, Addison-Wesley Publishing Company, Inc., 1992.

INITIAL DISTRIBUTION LIST

	No. of copies
1. Defense Technical Information Center..... 8725 John J. Kingman Rd. STE 0944 Ft. Belvoir, VA 22060 - 6218	2
2. Dudley Knox Library..... Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3. Professor James C. Emery, Code SM/Ey..... Naval Postgraduate School Monterey, CA 93943-5000	1
4. Professor Magdi N. Kamel, Code SM/Ka..... Naval Postgraduate School Monterey, CA 93943-5000	1
5. Evangelos P. Marinos..... 31 Perikleous Str., Aegaleo 12244, Athens, Greece.	2
6. Panagiotis P. Lymberis..... 580 Irving Ave. #A, Monterey, CA 93940.	1