NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

A DATABASE SYSTEM

FOR

THE MANAGEMENT OF NUMERICAL EXPERIMENTS

by

Maria Yuen-Ling So

Thesis

submitted to the

School of Graduate Studies and Research

in partial fulfillment of the

requirements for the degree of

Masters

of

Computer Science

Department of Computer Science

University of Ottawa

July 1987

ISBN 0-315-40743-3

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ACKNOWLEGEMENTS

I wish to express my deep gratitude and sincere appreciation to my thesis supervisor, Professor Louis G. Birta, for his valuable time, guidance, patience and advice during the research and preparation of this thesis. His generous financial support provided throughout the duration of my research is gratefully acknowledged.

I am grateful to all my professors, especially Professor Robert L. Probert for his help and encouragement.

I would also like to extend my thanks to many friends, colleagues, and the staff of the Department of Computer Science, University of Ottawa, for their support and assistance.

ABSTRACT

Experimentation with numerical software typically generates large amounts of data. The usefulness of this data depends very much on the convenience with which it can be examined, summarized and/or tabulated. In this thesis, we outline the design of a generalized Numerical Experiments Management System (NEMS) that provides a framework for managing the large amount of relevant data that are typically associated with numerical studies. This includes the management of both the parameter values that are used in the experiments and the results that are produced. The work is based on a model of the problem solving environment which consists of three types of entities called solution engines, problem instances and experiments. The system environment for NEMS consists of a personal computer which runs NEMS and provides the database function, and a mainframe computer which carries out the experiments. The necessary interconnection mechanism is assumed to be available. The organization of NEMS and its functional capabilities are described in this thesis.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION FOR THE THESIS

In this thesis, the problem that we consider is that of developing a convenient and effective means for managing the data that is associated with extensive sets of numerical experiments. Such experiments could include investigations with linear and nonlinear programming codes, dynamic system studies based on the use of packages for the solution of ordinary or partial differential equations, the use of engineering design software, and simulation studies in general.

Such experimentation can be carried out from either of two perspectives; namely, that of the software developer or that of the end-user. For a software developer, the objective is generally to develop a program module that implements some algorithmic procedure (or procedures) intended for solving some specific class of problems; e.g. nonlinear optimization. The effectiveness of the module (or more properly, the underlying procedure) is then evaluated by testing it using numerous representative problems from the target class. On the other

hand, experimentation for the end-user usually revolves around a specific problem which needs to be solved. The problem frequently involves parameters which have a range of values that must be investigated. Alternately (or perhaps simultaneously) it may be desired to explore different solution strategies for the problem, as contained in several distinct software packages.

The common feature in both these contexts is the need to carry out a relatively large number of numerical experiments. This naturally leads to a rapid accumulation of large amounts of data whose usefulness depends very much on the convenience with which it can be examined, summarized and/or tabulated. The design and development of an environment for carrying out this data management function is the purpose of this thesis project.

We note finally that although the presentation of this thesis is at a general level, the implementation is oriented around : (a) a PC running MS-DOS and the dBASE III system; (b) a mainframe computer running VM/CMS. This orientation emerges at several places in the presentation where the discussion requires specific detail.

1.1.1 PROBLEM FORMULATION

A fundamental constituent of the computer-based experimentation with which the thesis is concerned, is the notion of a "solution engine" (SE). An SE is a unit of program code which implements one (or possibly several) solution procedure(s) designed to solve problems of a particular class. Representative examples of SE's are :

(a) a linear programming package which generates a solution to linear programming problems,

(b) a program package which contains a collection of procedures for solving ordinary differential equations (some of the procedures in the package may be better suited to solving certain types of problems; e.g. stiff equations)

(c) a simulation package specifically designed to accommodate queuing models.

The solution engine represents a computational work-horse : a prerequisite for its activation is a problem whose presentation conforms to predefined compatibility requirements. Such a problem presentation is typically in the form of code and/or data. It is an identifiable unit which we refer to in this thesis as a "problem instance" (PI).

The notions of an SE and a PI as outlined above, are very general and, in fact, admit situations that are outside the intended scope of this thesis project. For example, it would be possible to identify a payroll program (system) as an SE, in which case the associated PI could be an employee file containing employment data for some particular week. This example is not entirely consistent with the SE/PI specifications because the PI lacks a structural attribute that could be regarded as invariant; i.e. the PI is "all data". More specifically, our interpretation of a PI in this thesis will always incorporate a loose notion of a "model".

Both the solution engine and the problem instance, typically incorporate parameters and the main concern of the experimentation activity is with studying the results produced when various parameter sets are selected. This process can generate large amounts of data which rapidly become unmanageable if managed in the form of printed output. Furthermore without software support the typical requirement of carrying comparative analysis becomes exceedingly awkward.

The generalized Numerical Experiments Management System (NEMS) provides a framework for managing the large amount of relevant data that are typically associated with numerical studies. This includes

management of parameter values for both the solution engine (or engines) being used and the problem instances being studied, together with the results that are produced in the experiments.

1.1.2 PREVIOUS WORK

Previous discussions in the literature about the management of numerical experimentation have occurred within the context of simulation activities. The work of Ören and Zeigler [1], for example, suggests several file types that would be of value in a well-developed simulation environment. The Simulation Data Language (SDL) developed by Standridge incorporates a database management system specifically designed to interface with simulation software and to provide a variety of capabilities in terms of the presentation of simulation results. More recently Korn [4] describes the DESCTOP system which integrates a simulation language environment with a file management system that maintains both program code and simulation results. The work described in this thesis shares several similarities with the functional characteristics of these previous developments. Its distinctiveness derives both from the generalized framework within which it is carried out and its goal of

providing a general purpose manager both for the execution of numerical experiments and for the data (i.e. results) that such experimentation produces.

1.2 PREVIEW OF THE THESIS

This thesis outlines the design and implementation of NEMS (Numerical Experiments Management System), a database system for the management of numerical experiments. Chapter 2 discusses the program design philosophy of the system, including the system environment, NEMS system functions, program design of NEMS and user requirements. Chapter 3 gives the implementation aspects for the four NEMS system functions : "Create", "Execute", "Query" and "Delete". Chapter 4 presents an example of the use of NEMS. Chapter 5 includes some concluding remarks and possible extension that would enhance the usefulness of the current version of NEMS.

CHAPTER 2

OVERVIEW OF THE DESIGN AND FUNCTIONS OF NEMS

2.1 CONCEPTUAL ENVIRONMENT

The NEMS software manipulates three classes of objects, namely, solution engines (SE's), problem instances (PI's), and experiments. In the following sections, we outline the pertinent features of each of these object classes.

2.1.1 SOLUTION ENGINES

We regard a solution engine as a piece of program code that resides on some mainframe computer to which NEMS has access. This code carries out a specific problem solving task (e.g. solution of the linear programming problem or the solution of a system of partial differential equations). A solution engine is characterized by two groups of attributes, namely, defining attributes and operational attributes.

The defining attributes give information about the solution engine's creation and underlying structure. These attributes are invariant (once specified, they are never altered). A detailed discussion of the defining attribute is given in Section 2.4.

The operational attributes relate either to the way in which the solution engine carries out its problem solving task or to the results which it generates. Typically one or more of these attributes changes in value from one use of the solution engine to another. These operational attributes fall into two categories; namely, parameters and variables.

Typically, a solution engine is designed to function under a range of circumstances; e.g. various sizes of some input data array, various values of some termination parameter or various values of a parameter that controls solution accuracy. We refer to attributes of this type as "parameters". A distinguishing feature of a parameter is that it must be assigned a value before the engine can be executed. We note also that parameters can have a type which is either numeric or character. Furthermore, they can be structured as either scalars or vectors.

When a solution engine is executed, values are acquired by some special set of internal variables. The problem solution that is sought is typically contained among these values. These special variables that characterize the solution generated by the solution engine are called output variables or simply "variables". Like parameters, solution engine variables can have a type which is either character or numeric and they can be structured as either scalars or vectors. In addition, however, we

recognize another permissible structure for solution engine variables which we call a cluster-list (clist). A cluster-list can be regarded as an ordered set of vectors. An application for this concept would be in characterizing the solution trajectory of a set of ordinary differential equations. Suppose, for example, that a problem of this class contains a set of n first order differential equations. From the problem solver's perspective, the solution will likely consist of the set of vectors

$$(t(k), x1(k), x2(k), \ldots\ldots , xn(k)), \quad k = 1,2, \ldots\ldots , m$$

The first entry in each such vector is a value of the problem independent variable and the remaining entries are solution values at t(k). Such a collection of vectors would be regarded in NEMS as a cluster-list.

2.1.2 PROBLEM INSTANCES

A solution engine is designed to solve problems of a particular class. Each specific problem of the target class is called a problem instance (PI), and exists as a piece of program code. Like a solution engine, a problem instance is characterized by two groups of attributes called defining attributes and operational attributes. The nature and role of these two categories of attributes parallels the equivalent concepts as applied to solution engines.

In the context of NEMS, we regard each problem instance as being associated with a unique solution engine, i.e. the problem instance can only be "solved" using some particular solution engine. The associated solution engine is referred to as the problem instance's "master" and correspondingly, one of the defining attributes of each problem instance identifies this particular solution engine.

In many circumstances, there exists some parameters for a solution engine which it shares with every problem instance that is "owned" by it. Consider for example, a solution engine designed to carry out a nonlinear function minimization task. When applied to any particular problem, a fundamental value that is likely required by the solution engine is the size of the problem; i.e., the number of free parameters. Clearly this particular attribute is also a fundamental characteristic of the problem itself.

Such special parameters could be regarded as "belonging" to either the solution engine or the problem instances owned by the solution engine. In the design of NEMS, the former alternative was selected as the means of handling such circumstances. To distinguish these special parameters, they are referred to as "reflected" parameters. Thus, reflected parameters occur in the specification for a solution engine, but

they have their existence within the context of every problem instance owned by that solution engine; i.e. they are inherited by these problem instances.

2.1.3 EXPERIMENTS

An experiment is a relationship (or pairing together) of a solution engine and a (compatible) problem instance. The implicitly assumed effect of the union is the generation, by the SE, of a solution to the PI. This action, in particular, yields specific values for the variables of both the SE and the PI. Furthermore, a prerequisite of the action is the assignment of values to the parameters of both the SE and the PI. The fundamental task of NEMS is to capture the parameters and variables associated with an experiment and to organize this data in a meaningful (i.e. useful) way for the user.

2.2 SYSTEM ENVIRONMENT

The system environment of NEMS consists of a personal computer and a mainframe. In the following sections, we discuss each of the hardware and software environments of the NEMS system in detail.

2.2.1 HARDWARE ENVIRONMENT

The design of the NEMS system has been based on an interconnected personal computer/mainframe configuration. In addition, the necessary communication link between these two components is assumed to exist.

The primary purpose of the mainframe is to provide the computing power for the experiments that are to be carried out within the context of the system. In addition, the program code for all SE's and PI's that are known to the system, is assumed to reside in appropriate files on the mainframe. The personal computer which carries out the data management task uses some resident database management system. There may, however, exist circumstances where the PC itself is sufficiently powerful to handle the anticipated computational workload and in such cases, a simplified system configuration without the mainframe would be possible.

The basis for this configuration stems from the following considerations:

(a) The program code for the SE(s) may be extensive and its execution during the problem-solving phase may require substantial CPU power in order to achieve solutions within a reasonable time period. These features could preclude the practicality of a wholly PC-based

environment.

(b) Flexible and powerful database management systems are widely available for PC's and hence it is both feasible and convenient to carry out the browsing/evaluation activities of the NEMS concept in a low-cost PC environment.

2.2.2 SOFTWARE ENVIRONMENT

The software component of NEMS is resident on the PC. The objects manipulated by NEMS are SE's, PI's and experiments (note that experiments are relationships between SE's and PI's). The basic functional requirement of NEMS is to provide a database management capability within the context of a numerical experimentation activity. Inasmuch as general database management software for PC's is widely available, the approach chosen in the development of NEMS is to create around such an available system, a superstructure designed to accommodate the specialized needs for the experimentation activity.

Among the three different types of database models, namely, hierachical, network and relational, we have chosen to use the relational model for the implementation of this thesis project. In a relational database, all data is viewed as being stored in tables. Each of these tables

closely resembles a conventional sequential file, with rows of the table corresponding to records of the file and columns corresponding to fields of the record. These tables are commonly referred to as relations, and each row of the relation is normally referred to as a tuple.

The principal advantage of the relational model is its relative ease of use. This is a direct consequence of the natural appeal of the tabular format for data representation. Also, good data independence can be achieved more easily with the relational structure than with the hierachical or network structures. If the database is in a normalized form with data independence in the software, the data can be restructured, and the database can grow without the need to rewrite application program code.

There is a wide selection of database systems available for use on PC's. We have chosen dBASE III as the underlying database system in the implementation of this thesis project. dBASE III provides powerful commands to perform database creation, data entry, updating, query, etc. and it also has a complete set of programming language commands which enables the programmer to write programs to perform a wide variety of data manipulation tasks. Another outstanding feature of dBASE III is that it can conveniently interface with algorithmic programming languages

like TURBO PASCAL with which it exchanges data using conventional text files.

In the context of this thesis project, the main shortcoming of dBASE III relates to its restricted support of numeric data. dBASE III supports both integers and decimal numbers. However, the representation of decimal numbers is limited to a maximum of 15 digits. Furthermore, dBASE III does not support numeric data expressed in scientific notation, and its set of available mathematical functions is rather limited.

In the anticipated environment for NEMS, it is highly likely that much of the data will be expressed in scientific notation. Hence, in the implementation, we have chosen to store all numerics as character strings. To deal with the problems of validation, comparison, and other manipulations of the numeric data, we must therefore branch out to the "richer" environment of an algorithmic programming language (in this case, we have chosen TURBO PASCAL). Fortunately, the interface between dBASE III and other algorithmic languages is clean and easy to use, and this has considerably simplified the solutions to the problems encountered with numeric data.

2.3 NEMS SYSTEM MODES

NEMS has four system modes which we refer to as : create, execute, query and delete. In the create mode, the user can create the existence of SE's and PI's. The user can activate and experiment with an SE and PI in the execute mode. In the query mode, the user can browse through the data that has been accumulated in the NEMS database. The user can also delete the existence of SE's and PI's in the delete mode. The four NEMS system modes are discussed further in the following sections.

2.3.1 THE CREATE MODE

In the create mode, the user is provided with the means to create the existence of both solution engines and problem instances. This activity takes place as a dialog during which the relevant specifications are obtained from the creator. The basic steps in this dialog are summarized in Figure 2.1.

The user is prompted for the defining attributes of the solution engine or problem instance, depending on which entity he is creating. He enters the name of the solution engine/problem instance, the name of the owner, numeric precision of the solution engine/problem instance, and

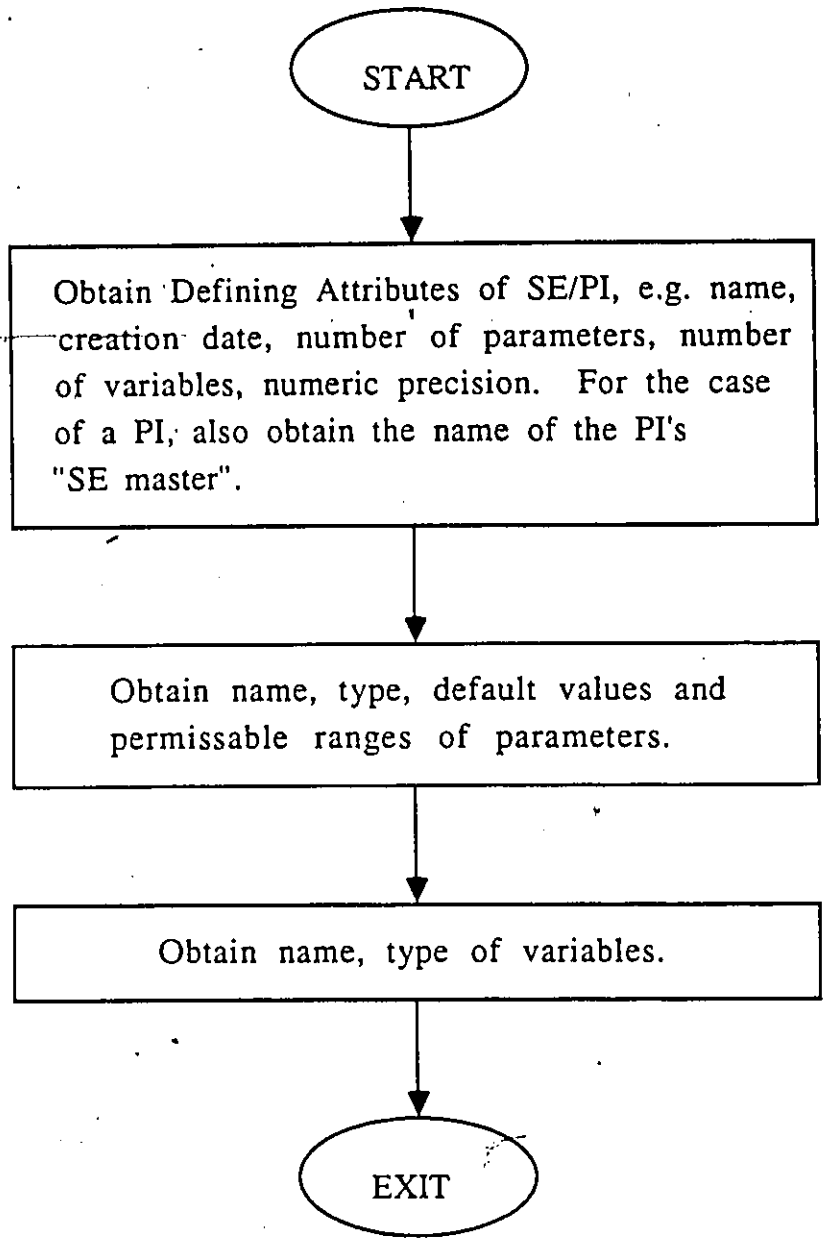


Figure 2.1

Basic Steps of the Create Mode.

the number of the parameters and variables. For the case of a problem instance, a user must also specify the SE "master" to be used with the problem instance. (Recall that each problem instance can only be used with a particular solution engine, although a solution engine can be the "master" of many problem instances).

After entering this information, the user is prompted for information about the parameters. He enters the name of each parameter, its type, default value and permissible range. For the case of vector parameter, the user specifies the maximum length of the vector and gives relevant information about its active length (i.e. its actual length during execution). The maximum length of a vector parameter is invariant; but its active length is likely to vary from experiment to experiment. NEMS provides several options for handling the specifications of active length; namely, the active length can be fixed at creation time, or be user-assignable at execution time, or be specified via the value of another scalar parameter, or via the active length of another vector.

The user is also prompted for information relating to the variables. This, in particular, involves specifying the name and type of each of the variables (the notions of default value and permissible range do not apply to variables). The length of a vector variable is handled the same

way as vector parameters. For a clist variable, the user specifies its maximum size and gives relevant information about its active size.

After these specifications are obtained, NEMS assigns to the solution engine/problem instance a NEMS internal identifier. The system then creates all the relations required to maintain the relevant information. NEMS adds the solution engine/problem instance to an appropriate library and creates the appropriate additional relations that are necessary for storing values for parameters and variables.

A solution engine/problem instance is defined only once. After its creation, the user can use the solution engine/problem instance in an experiment, or browse through the data associated with it.

2.3.2 THE EXECUTE MODE

The execute mode corresponds to the carrying out of an experiment and thus to the creation of an instance of an "Experiment object". During a preliminary set-up dialog with the user, various specifications are acquired. In particular, both the solution engine and problem instance to be used in the experiment must first be identified.

The active values of each of the parameters of both the solution engine and the problem instance must then be obtained from the user.

These values are checked for consistency with the prescribed range values.

As previously outlined, various options are available for establishing the active length of a vector parameter. For each such parameter (in either an SE or PI) the desired option is stored among the defining attributes of the SE or PI (as appropriate). During the set-up dialog of the execution mode, explicit input from the user relating to the active length of a particular vector is required only in one circumstance. This is the case where the active length has been specified to be "user-assignable at execution time".

In addition to active length specification for parameter vectors. The user may also be required to provide active length specifications for variable vectors within the SE and/or PI. This need will depend on the nature of the stored option that is specified in this regard. Only for the case where the active length has been specified as "user-assignable at execution time", will explicit user input be required.

Similarly, there may be a requirement for the user input of a value for the active size of some clists within the SE and/or PI. This also will depend on the active size option that was specified during the creation phase.

The set-up dialog continues until NEMS acquires values for all parameters, active lengths and sizes that are needed for the experiment that is being initiated.

The data acquired from the user is placed into appropriate NEMS relations. In addition, a "set-up" file is generated which incorporates the parameter values and other pertinent data (e.g. the specified SE and PI to be used in the experiment). This file provides the data necessary for the setting up of the mainframe-resident SE and PI that are to be used in the experiment.

Communication with the remote mainframe is then initiated and the user transfers the set-up file from the PC to the mainframe. The "NEMS-remote" procedure (a module resident on the mainframe) is then activated by the user and this results in the reading of the set-up file and the subsequent execution of the prescribed solution engine/problem instance combination. After completion of the experiment, the results (as written to a pre-determined file) are transferred back to the PC where NEMS extracts values for the solution engine/problem instance variables in the appropriate relations. The basic steps associated with the execute mode are given in Figure 2.2.

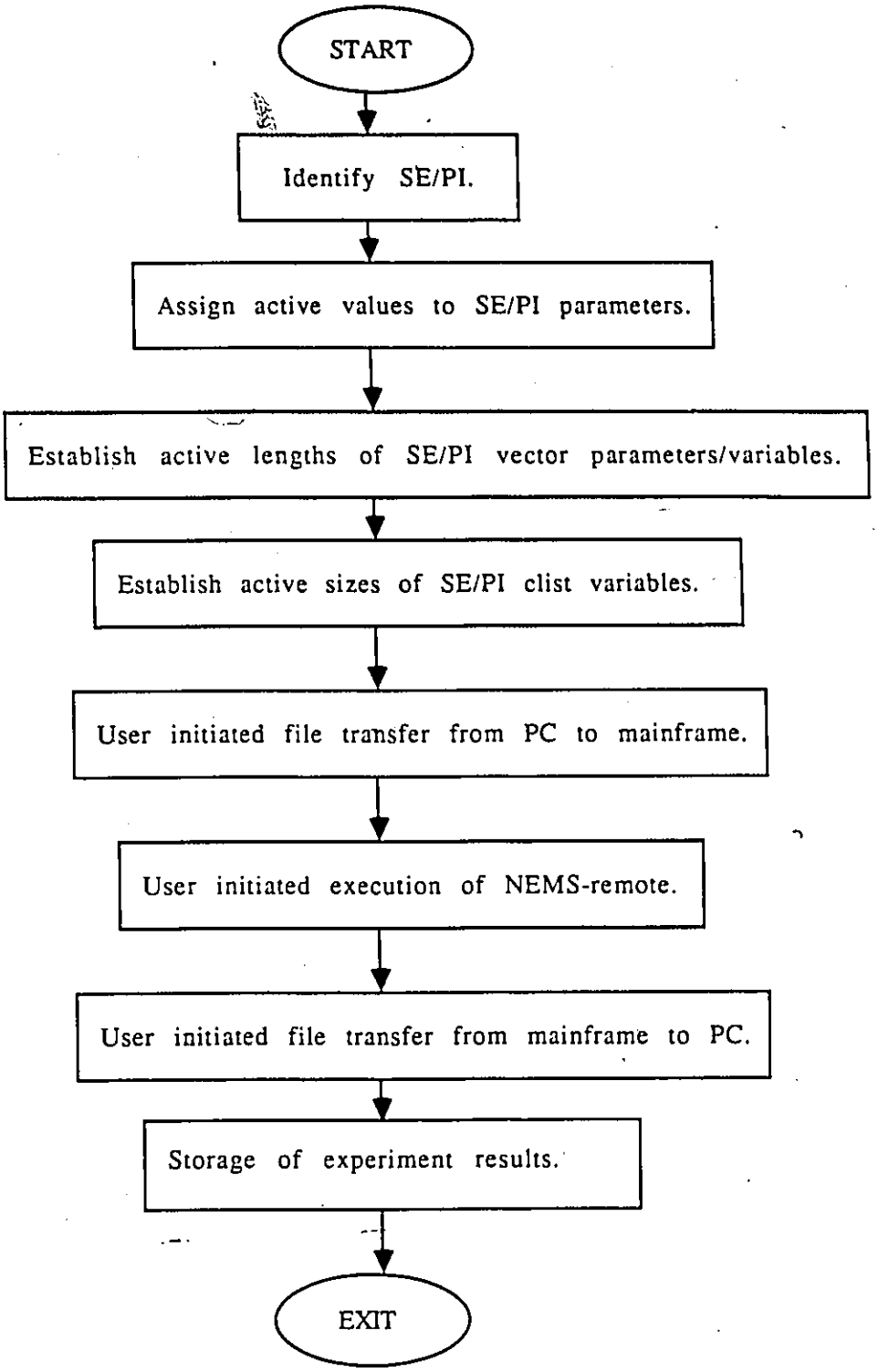


Figure 2.2

Basic Steps of the Execute Mode.

2.3.3 THE QUERY MODE

It is in the query mode that the user is able to browse through the data that has been accumulated in the NEMS database. The query subsystem is menu-driven and provides a selection of queries which are of particular relevance to the user. One of the basic objectives of the query mode is to provide a means for conveniently displaying the values of the parameters and variables in the experiments associated with a solution engine and/or a problem instance.

In the query mode, the user can, in particular, list the collection of solution engines and problem instances in the NEMS library. He can examine the defining attributes, such as owner's name, creation date, numeric precision, etc., of a solution engine or problem instance. The user can also obtain specification information about the parameters and variables, such as their names, types, default values and permissible ranges. Experiments carried out with a solution engine or problem instance can also be examined. To do so, the particular solution engine and/or problem instance that is of interest must first be "designated".

After designation, the user can, for example, display all the problem instances associated with a designated solution engine. The user can also "refine" what he wishes to see by "masking" parameters and/or variables

that are not of particular interest. After the masking operation, the masked parameters/variables do not appear in any subsequent display. The user can also "specialize" ranges of values for parameters and variables that he wishes to consider. After specialization, only experiments which satisfy the range conditions are displayed. Note that both the masking and specialization operations can be re-initialized, that is, the user can always unmask the masked parameters/variables, and eliminate range specifications.

This query subsystem is menu-driven and the user moves from one menu to another. This query subsystem provides a convenient means to browse through the NEMS database and examine the accumulated data relating to SE's, PI's and experiments. A representation of the various stages associated with the query mode is given in Figure 2.3.

2.3.4 THE DELETE MODE

In the delete mode, the user can delete the existence of an SE or a PI. The process of deleting an instance of an SE or a PI is very straightforward. The user is simply prompted for the name of the SE or PI that he wishes to delete. After this identification, NEMS performs the necessary actions to delete the SE or PI and all the information relating to

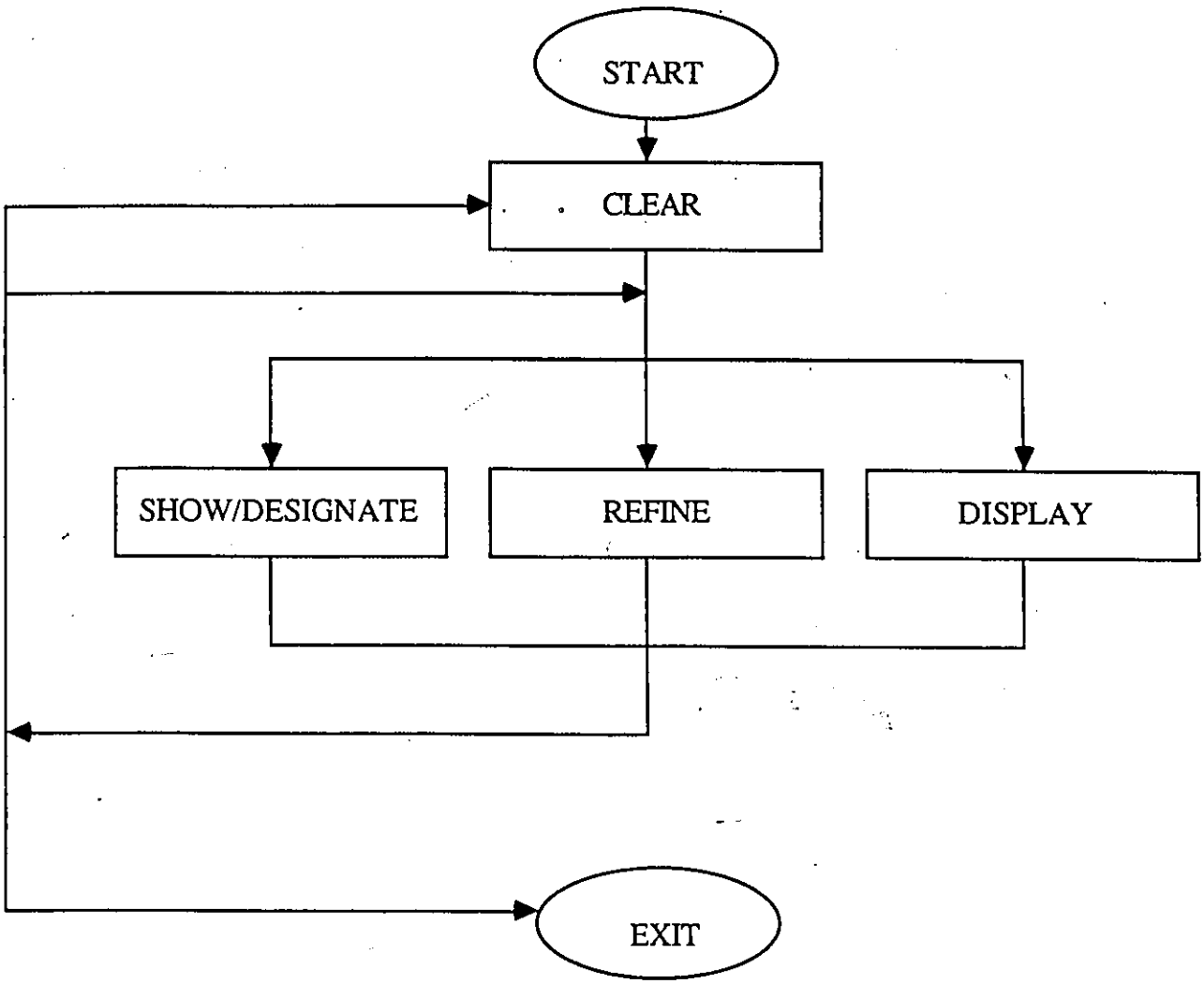


Figure 2.3

Basic Steps of the Query Mode.

it. Figure 2.4 gives a representation, from the user's perspective, of the basic steps involved in the deletion process. The deletion operation can result in extensive activity within NEMS, particularly in the case of the deletion of an SE. In such a case, NEMS must delete not only every experiment that is associated with the SE, but also delete every PI that has this SE as its master. The situation can be aggrevated when the SE and PI's have vector parameters or variables associated with them.

2.4 STORAGE STRUCTURES

The data relating to the three underlying entities : SE's, PI's and experiments, are organized as a set of dBASE III relations. This collection of relations is discussed in detail in the following sections.

Figure 2.5 gives a global view of these various relations that can exist in NEMS as well as their structural relationships. In this Figure a single arrowhead indicates a one-to-one relationship, while a double arrowhead indicates one-to-many. This Figure provides a unifying perspective for the discussion which follows.

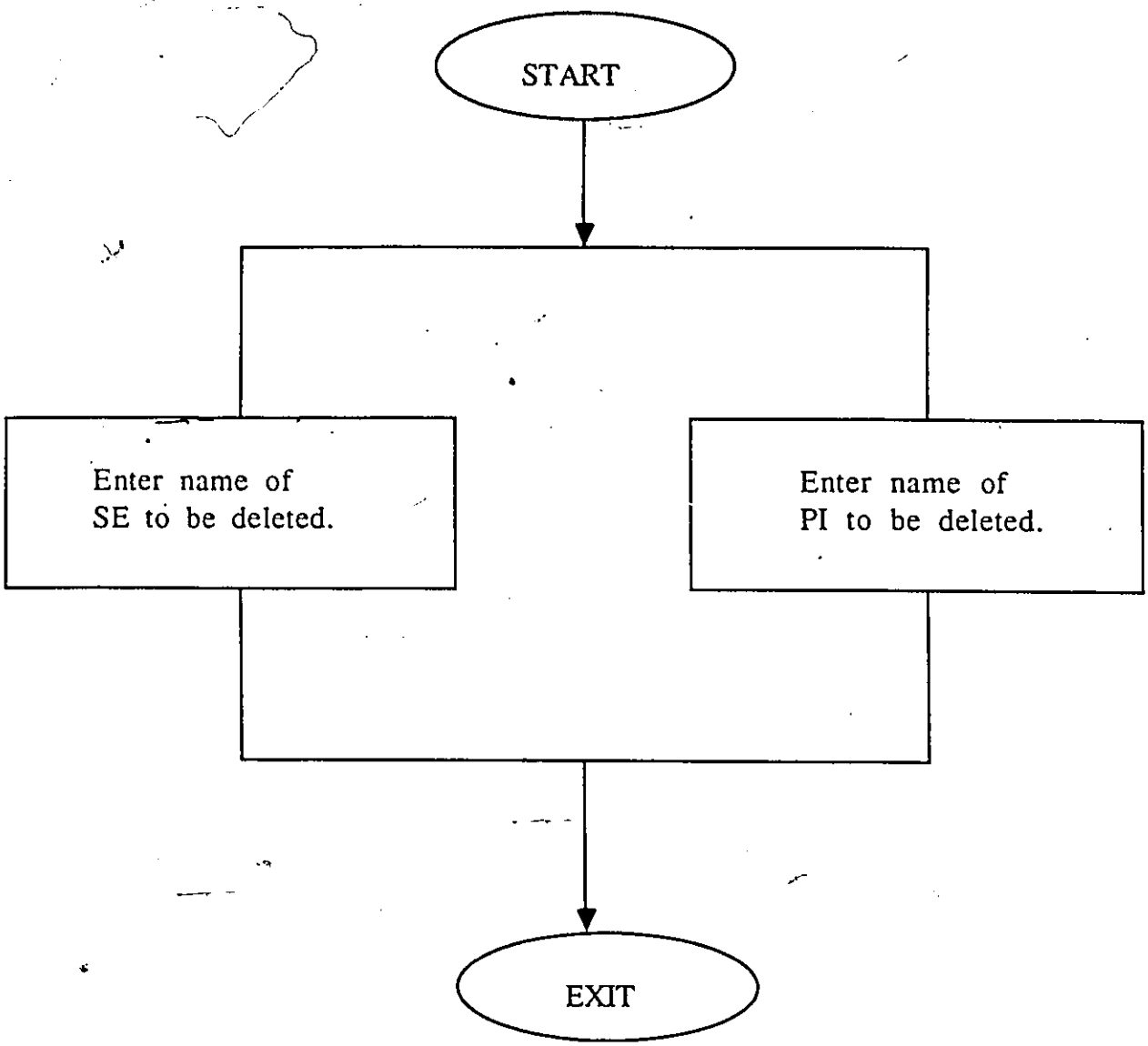


Figure 2.4

Basic Steps of the Delete Mode.

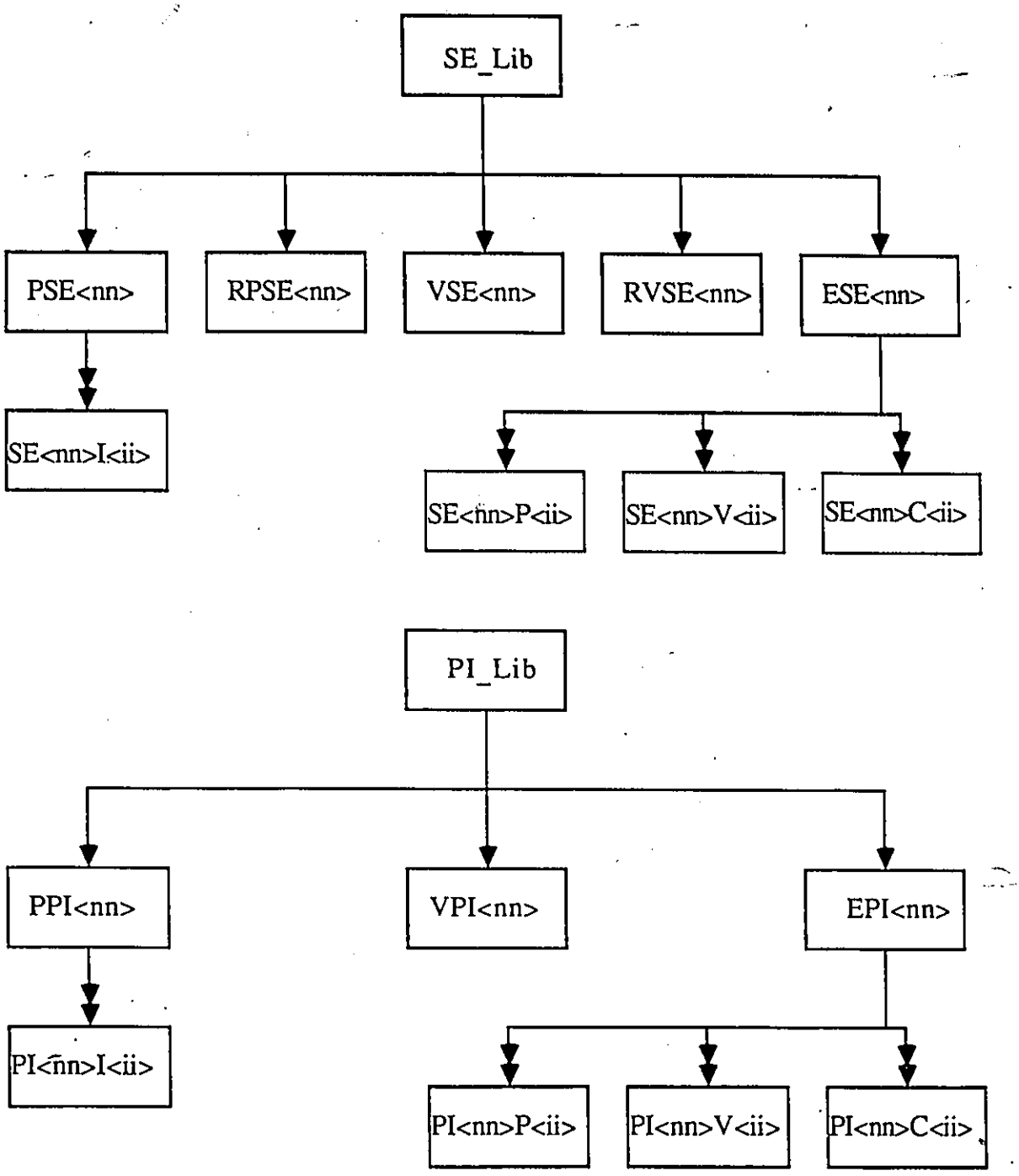


Figure 2.5

Interrelationships among the NEMS Relations.

2.4.1 STORAGE OF SE INSTANCES

Each SE defined to the system is described via a set of "Defining attributes". This characterizing data for each SE is maintained as a tuple in a relation called SE_Lib. The attributes of SE_Lib are summarized in Table 2.1 .

2.4.2 STORAGE OF SE PARAMETER SPECIFICATIONS

As outlined in Section 2.1.1 , each SE known to the system is associated with a distinctive set of parameters whose values control the operational characteristics of the engine. The specifications for each of the parameters associated with the SE whose internal identifier is SE<nn> are maintained as tuples in a table called PSE<nn>. The attributes of PSE<nn> are summarized in Table 2.2 and the following explanatory notes relate to Table 2.2.

NOTE 2.1

For the case of a vector parameter, a user must specify the vector's maximum length at creation time, and this maximum length is invariant. A vector's active length (which applies at execution time) may be invariant; however it is more likely that it will vary from experiment to

ATTRIBUTE	TYPE	SIZE	DATA
SE_NAME	CHARACTER	15	User specified name for the SE.
SE_CODE	CHARACTER	4	NEMS internal identifier for the SE*.
CR_DATE	DATE	8	Creation date of the SE.
OWNER	CHARACTER	15	Name of the owner of the SE.
NUM_PAR	INTEGER NUMERIC	2	Number of parameters.
NUM_RPAR	INTEGER NUMERIC	2	Number of reflected parameters.
NUM_VAR	INTEGER NUMERIC	2	Number of output variables.
NUM_RVAR	INTEGER NUMERIC	2	Number of reflected output variables.
PRECISION	CHARACTER	1	Numerical precision (Single/Double).
NUM_USAGE	INTEGER NUMERIC	4	Total number of usages of the SE.
EXEC_NAME	CHARACTER	15	Name of the Exec file which executes the SE.

Table 2.1

Attributes of the Relation SE_Lib.

* The internal identifier for SE's are of the form SE<nn>. The notation <nn> is used to represent an arbitrary two digit identifier.

ATTRIBUTE	TYPE	SIZE	DATA
PAR_NAME	CHARACTER	15	Parameter name.
PAR_CODE	CHARACTER	3	NEMS internal identifier for the parameter.
PAR_TYPE	CHARACTER	9	Parameter type. [Possible values are : Scalar, Vector, Character.]
N_TYPE	CHARACTER	7	Numeric type of parameter. [Relevant only to the case where PAR_TYPE is not Character.]
LENGTH	INTEGER NUMERIC	2	Maximum length of vector. [Relevant only to the case where PAR_TYPE is Vector.]
VIA	CHARACTER	15	A parameter/variable name through which the active length of a vector is obtained. [Relevant only to the case where PAR_TYPE is Vector.]
LEN_CODE	CHARACTER	2	Active length code. [Relevant only to the case where PAR_TYPE is Vector. See Note 2.1.]
DEFAULT	CHARACTER	15	Default value. [See Note 2.2 for the case where PAR_TYPE is Vector.]
LO_BOUND	CHARACTER	15	Lower bound. [See Note 2.2 for the case where PAR_TYPE is Vector.]
UP_BOUND	CHARACTER	15	Upper bound. [See Note 2.2 for the case where PAR_TYPE is Vector.]

Table 2.2

Attributes of the Relation PSE<nn>.

experiment. Thus some degree of flexibility is required. NEMS provides several options for handling the specifications of active length; namely, the active length can be fixed at creation time, or be user-assignable at execution time, or be specified via the value of another scalar parameter, or via the active length of another vector. The value of the LEN_CODE attribute gives information on how the active length is to be obtained. The possible values of this LEN_CODE attributes are : n, PS, PV, X. NEMS interprets the value stored in the LEN_CODE attribute and carries out the appropriate actions to obtain the correct active length for the vector parameter when the solution engine is activated. Table 2.3 gives the interpretation for each possible value of the LEN_CODE attribute.

NOTE 2.2

A vector parameter can have a default value, as well as upper and lower bounds, for each of its components. Since only scalar values can be stored in dBASE III relations, it is necessary to create a supplementary relation to store these attributes for each vector parameter. Assume we have a solution engine whose SE_CODE is SE<nn>, which has a vector parameter whose PAR_CODE is P<ii>; then the information for the components of this vector parameter is stored in a relation called

LEN_CODE	INTERPRETATION
n	The active length of the vector in question is explicitly given as n.
PS	The active length of the vector in question is the same as the value of the SE scalar parameter stated in the VIA attribute.
PV	The active length of the vector in question is the active length of SE vector parameter stated in the VIA attribute.
X	The active length of the vector in question is provided by the identifier stated in the VIA attribute. This identifier is assumed to be a scalar or vector parameter, or a vector variable of the particular PI used with the SE during an experiment.

Table 2.3

Interpretation of Active Length Codes.

(SE Vector Parameter)

SE<nn>I<ii>. Furthermore, the value assigned to the DEFAULT attribute of this vector parameter in the relation PSE<nn> is the name of the relation which stores the information. Similarly, the value assigned to the UP_BOUND and LO_BOUND attributes of this vector parameter is also SE<nn>I<ii>. Further discussion on the relation SE<nn>I<ii> is given in Section 2.4.4.

2.4.3 STORAGE OF SE REFLECTED PARAMETER SPECIFICATIONS

The reflected parameters of an SE have a special role as outlined in Section 2.1.2. The specifications for each of the reflected parameters associated with the SE whose internal identifier is SE<nn> are maintained as a tuple in a table called RPSE<nn>. The attributes of RPSE<nn> are summarized in Table 2.4 and the following explanatory note relates to Table 2.4.

NOTE 2.3

The method for specifying the active length of a reflected vector parameter parallels that used in the case of the active length of a conventional (i.e. non-reflected) vector parameter. (See Note 2.1.) The value of the LEN_CODE attribute gives information on how the active

ATTRIBUTE	TYPE	SIZE	DATA
PAR_NAME	CHARACTER	15	Parameter name.
PAR_CODE	CHARACTER	3	NEMS internal identifier for the parameter.
PAR_TYPE	CHARACTER	9	Parameter type. [Possible values are : Scalar, Vector, Character.]
N_TYPE	CHARACTER	7	Numeric type of parameter. [Relevant only to the case where PAR_TYPE is not Character.]
LENGTH	INTEGER,NUMERIC	2	Maximum length of vector. [Relevant only to the case where PAR_TYPE is Vector.]
VIA	CHARACTER	15	A parameter/variable name through which the active length of a vector is obtained. [Relevant only to the case where PAR_TYPE is Vector.]
LEN_CODE	CHARACTER	2	Active length code. [Relevant only to the case where PAR_TYPE is Vector. See Note 2.3.]

Table 2.4

Attributes of the Relation RPSE<nn>.

length is to be obtained. The possible values of the LEN_CODE attribute are : n, PS, PV, X. Table 2.5 gives the interpretation for each possible value of the LEN_CODE attribute.

2.4.4 STORAGE OF SE VECTOR PARAMETER SPECIFICATIONS

As indicated in Note 2.2, the default value, as well as the upper and lower bounds of each of the component of a vector parameter are stored as a tuple in a relation called SE<nn>I<ii>, where SE<nn> is the internal identifier of the associated solution engine and P<ii> is the PAR_CODE of the vector parameter in question. The attributes of SE<nn>I<ii> are summarized in Table 2.6.

2.4.5 STORAGE OF SE VARIABLE SPECIFICATIONS

Each SE known to the system is associated with a distinctive set of variables whose values capture some portion of the experimental results associated with the use of the engine. (See Section 2.1.1.) The specifications for each of the variables associated with the SE whose internal identifier is SE<nn> are maintained as a tuple in a table called VSE<nn>. The attributes of VSE<nn> are summarized in Table 2.7 and the

LEN_CODE	INTERPRETATION
n	The active length of the vector in question is explicitly given as n.
PS	The active length of the reflected vector parameter in question is the same as the value of the SE reflected scalar parameter stated in VIA attribute.
PV	The active length of the reflected vector parameter in question is the same as the active length of the SE reflected vector parameter stated in the VIA attribute.
X	The active length of the reflected vector parameter in question is provided by the identifier stated in the VIA attribute. This identifier is assumed to be an SE scalar parameter or an SE vector parameter.

Table 2.5

Interpretation of Active Length Codes.

(SE Reflected Vector Parameter)

ATTRIBUTE	TYPE	SIZE	DATA
COMP_NO	INTEGER NUMERIC	4	Component number in question.
LO_BOUND	CHARACTER	15	Lower bound.
UP_BOUND	CHARACTER	15	Upper bound.
DEFAULT	CHARACTER	15	Default value.

Table 2.6

Attributes of the Relation SE<nn>I<ii>.

ATTRIBUTE	TYPE	SIZE	DATA
VAR_NAME	CHARACTER	15	Variable name.
VAR_CODE	CHARACTER	3	NEMS internal identifier for the variable.
VAR_TYPE	CHARACTER	9	Variable type. [Possible values are : Scalar, Vector, Character,Clist.]
N_TYPE	CHARACTER	7	Numeric type of variable. [Relevant only to the case where VAR_TYPE is not Character.]
LENGTH	INTEGER NUMERIC	2	Maximum length of vector (if VAR_TYPE is Vector) or maximum size of clist (if VAR_TYPE is Clist). [Relevant only to the case where VAR_TYPE is Vector or Clist.]
VIA	CHARACTER	15	The parameter/variable name through which the active length of a vector or the active size of a clist is obtained. [Relevant only to the case where VAR_TYPE is vector or Clist.]
LEN_CODE	CHARACTER	2	Active length code of a vector (if VAR_TYPE is Vector). Active size code of a clist (if VAR_TYPE is Clist). [Relevant only to the case where VAR_TYPE is Vector or Clist.] (See Note 2.4)

Table 2.7

Attributes of the Relation VSE<nn>.

following explanatory note relate to Table 2.7.

NOTE 2.4

The active length of a vector variable is established in a similar way to the active length of a vector parameter. (See Note 2.1.) The value assigned to the LEN_CODE attribute specifies how NEMS is to obtain the active length the vector variable. The possible values for the LEN_CODE attribute are : n, PS, PV, VV, X and Table 2.8 gives the interpretation for each of the possible values.

The maximum size of a clist variable is specified by the user at creation time, and this assigned value is invariant. However, a clist's active size, which applies at execution time, may vary from experiment to experiment. This active size can be either fixed at creation time, or user-assignable at execution time, or be specified via the value of another scalar parameter, or via the active size of another clist. The value of the LEN_CODE attribute gives information on how the active size is to be obtained. The possible values of this LEN_CODE attributes are : n, PS, X. NEMS interprets the value stored in the LEN_CODE attribute and carries out the appropriate actions to obtain the correct active size for the clist variable. Table 2.9 gives the interpretation for each possible value of

LEN_CODE	INTERPRETATION
n	The active length of the vector in question is explicitly given as n.
PS	The active length of the vector in question is the same as the value of the SE scalar parameter stated in the VIA attribute.
PV	The active length of the vector in question is the active length of SE vector parameter stated in the VIA attribute.
VV	The active length of the vector in question is the active length of SE vector variable stated in the VIA attribute.
X	The active length of the vector in question is provided by the identifier stated in the VIA attribute. This identifier is assumed to be a scalar or vector parameter, or a vector variable of the PI used with the SE.

Table 2.8

Interpretation of Active Length Codes.

(SE Vector Variable)

LEN_CODE	INTERPRETATION
n	The active size of the clist in question is explicitly given as n.
PS	The active size of the clist in question is the same as the value of the SE scalar parameter stated in the VIA attribute.
X	The active size of the clist in question is provided by the identifier stated in the VIA attribute. This identifier is assumed to be a scalar parameter, or a clist variable of the PI used with the SE.

Table 2.9

Interpretation of Active Size Codes.

(SE Clist Variable)

the LEN_CODE attribute of a clist variable.

2.4.6 STORAGE OF SE REFLECTED VARIABLE SPECIFICATIONS

As outlined in Section 2.1.2, the reflected variables of an SE have a special role. The specifications for each of the reflected variables associated with the SE whose internal identifier is SE<nn> are contained in a table called RVSE<nn>. The attributes of RVSE<nn> are summarized in Table 2.10 and the following explanatory note relate to Table 2.10.

NOTE 2.5

The active length of a reflected vector variable is obtained in a similar way to the active length of a vector parameter. (See Note 2.1.) The value of the LEN_CODE attribute of a reflected vector variable gives information about how to obtain the active length. Possible values for the LEN_CODE attribute are : n, PS, PV, VV, RV, X and Table 2.11 gives the interpretation for each of these possible values.

The active size of a reflected clist variable is obtained in a similar way to the active size of a clist variable. (See Note 2.4.) The LEN_CODE attribute of a reflected clist variable gives information about how to obtain the active size of a clist variable. Possible values are : n, PS, X and

ATTRIBUTE	TYPE	SIZE	DATA
VAR_NAME	CHARACTER	15	Variable name.
VAR_CODE	CHARACTER	3	NEMS internal identifier for the variable.
VAR_TYPE	CHARACTER	9	Variable type. [Possible values are : Scalar, Vector, Character,Clist.]
N_TYPE	CHARACTER	7	Numeric type of variable. [Relevant only to the case where VAR_TYPE is not Character.]
LENGTH	INTEGER NUMERIC	2	Maximum length of vector (if VAR_TYPE is Vector) or maximum size of clist (if VAR_TYPE is Clist). [Relevant only to the case where VAR_TYPE is Vector or Clist.]
VIA	CHARACTER	15	The parameter/variable name through which the active length of a vector or the active size of a clist is obtained. [Relevant only to the case where VAR_TYPE is Vector or Clist.]
LEN_CODE	CHARACTER	2	Active length code of a vector (if VAR_TYPE is Vector). Active size code of a clist (if VAR_TYPE is Clist). [Relevant only to the case where VAR_TYPE is Vector or Clist.] (See Note 2.5)

Table 2.10

Attributes of the Relation RVSE<nn>.

LEN_CODE	INTERPRETATION
n	The active length of the vector in question is explicitly given as n.
PS	The active length of the reflected vector variable in question is the same as the value of SE scalar parameter stated in the VIA attribute.
PV	The active length of the reflected vector variable in question is the same as the active length of the SE vector parameter stated in the VIA attribute.
VV	The active length of the reflected vector variable in question is the same as the active length of the SE vector variable stated in the VIA attribute.
RV	The active length of the reflected vector variable in question is the same as the active length of the SE reflected vector variable stated in the VIA attribute.
X	The active length of the reflected vector vaariable in question is provided by the identifier stated in the VIA attribute. This identifier is assumed to be an SE scalar parameter or an SE vector parameter.

Table 2.11

Interpretation of Active Length Codes.

(SE Reflected Vector Variable)

Table 2.12 gives the interpretation for each of the possible values.

2.4.7 STORAGE OF ACTIVE VALUES FOR SE PARAMETERS

The initialization of an experiment via NEMS begins with the specification of some particular solution engine. The parameters associated with that SE must then be assigned specific values (i.e. active values) which will be used during the experiment execution. Furthermore, the problem instance associated with the experiment must also be identified. Each usage of an SE is assigned a sequential identification number and, in addition, each NEMS experiment has a sequential identification number. These various characteristics of an experiment carried out with an SE whose internal identifier is SE<nn> are stored as a tuple in a table called ESE<nn>. Table 2.13 summarizes the attributes of ESE<nn> and the following explanatory note relates to Table 2.13. Note also that the values of the SE output variables resulting from the experiment are also stored in the table.

NOTE 2.6

For the case of a vector parameter, the active values for each of the vector components are stored in a supplementary relation called

LEN_CODE	INTERPRETATION
n	The active size of the clist in question is explicitly given as n.
PS	The active size of the clist in question is the same as the value of the SE scalar parameter stated in the VIA attribute.
X	The active size of the clist in question is provided by the identifier stated in the VIA attribute. This identifier is assumed to be a scalar parameter, or a clist variable of the PI used with the SE.

Table 2.12

Interpretation of Active Size Codes.

(SE Reflected Clist Variable)

ATTRIBUTE	TYPE	SIZE	DATA
SE_SEQ_NUM	INTEGER NUMERIC	4	SE usage number.
XID	INTEGER NUMERIC	4	NEMS experiment ID.
PI_CODE	CHARACTER	4	NEMS internal identifier for the PI used in the experiment. [See Note 2.6]
P01	CHARACTER	15	Value assigned to the parameter whose PAR_CODE is P01. [See Note 2.6]
P02	CHARACTER	15	Value assigned to the parameter whose PAR_CODE is P02. [See Note 2.6]
o o o	o o o	o o o	o o o
P<nn>	CHARACTER	15	Value assigned to the parameter whose PAR_CODE is P<nn>.
V01	CHARACTER	15	Value acquired by the output variable whose VAR_CODE is V01.
V02	CHARACTER	15	Value acquired by the output variable whose VAR_CODE is V02.
o o o	o o o	o o o	o o o
V<nn>	CHARACTER	15	Value acquired by the output variable whose VAR_CODE is V<nn>.

Table 2.13

Attributes of the Relation ESE<nn>.

SE<nn>P<ii>, where SE<nn> is the internal identifier of the SE and P<ii> is the PAR_CODE of the vector parameter. In such a case, the "value" stored for the parameter P<ii> is the relation name SE<nn>P<ii>. Further details about the relation SE<nn>P<ii> are given in Section 2.4.8.

2.4.8 STORAGE OF ACTIVE VALUES FOR SE VECTOR PARAMETERS

Suppose we have a solution engine whose internal identifier is SE<nn>, which has a vector parameter whose PAR_CODE is P<ii>. When NEMS initiates an experiment, the active value assigned to each of the vector components ia entered as a tuple in a table called SE<nn>P<ii>. (See Note 2.6.) The attributes of SE<nn>P<ii> are summarized in Table 2.14.

2.4.9 STORAGE OF VALUES FOR SE VARIABLES

As outlined in Section 2.1.1, each SE known to the system is associated with a set of variables. These variables capture a portion of the results generated from the execution of an experiment. The values of the variables for a solution engine whose internal identifier is SE<nn>, are stored in the same table which stores the active values for the SE parameters; namely, in the table ESE<nn> whose attributes are

ATTRIBUTE	TYPE	SIZE	DATA
SE_SEQ_NUM	INTEGER NUMERIC	4	SE usage number.
ACT_COMP	INTEGER NUMERIC	15	Number of active components.
C01	CHARACTER	15	Value assigned to component number 1 for the experiment.
C02	CHARACTER	15	Value assigned to component number 2 for the experiment.
o o o	o o o	o o o	o o o
C<nn>	CHARACTER	15	Value assigned to component number <nn> for the experiment.

Table 2.14

Attributes of the Relation SE<nn>P<ii>.

summarized in Table 2.13.

2.4.10 STORAGE OF VALUES FOR SE VECTOR VARIABLES

Suppose we have a solution engine whose internal identifier is SE<nn>, which has a vector variable whose VAR_CODE is V<ii>. When NEMS completes an experiment, the value obtained for each of the vector components is inserted in a table called SE<nn>V<ii> whose attributes are summarized in Table 2.15. Note that the "value" stored for such a vector variable in the relation ESE<nn> (see Table 2.13) is simply the relation name SE<nn>V<ii>.

2.4.11 STORAGE OF VALUES FOR SE CLIST VARIABLES

Suppose we have a solution engine whose internal identifier is SE<nn>, which has a clist variable whose VAR_CODE is V<ii>. When NEMS completes an experiment, the data values of the clist variable are stored in a file in the mainframe. However, the total number of data points in this clist variable, and the name of the file (in the mainframe) which stores all these data values are maintained in a relation called SE<nn>C<ii>. The attributes of SE<nn>C<ii> are summarized in Table 2.16. Whenever a

ATTRIBUTE	TYPE	SIZE	DATA
SE_SEQ_NUM	INTEGER NUMERIC	4	SE usage number.
ACT_COMP	INTEGER NUMERIC	15	Number of active components.
C01	CHARACTER	15	Value acquired by component number 1 from the experiment.
C02	CHARACTER	15	Value acquired by component number 2 from the experiment.
o o o	o o o	o o o	o o o
C<nn>	CHARACTER	15	Value acquired by component number <nn> from the experiment.

Table 2.15

Attributes of the Relation SE<nn>V<ii>.

variable, V<ii>, is a clist, the "value" stored for it in the relation ESE<nn> (see Table 2.13) is the relation name SE<nn>C<ii>.

2.4.12 STORAGE OF PI INSTANCES

Each PI defined to the system is associated with a set of "Defining attributes". This characterizing data for each PI is maintained in a relation called PI_Lib. The attributes of PI_Lib are summarized in Table 2.17.

2.4.13 STORAGE OF PI PARAMETER SPECIFICATIONS

As outlined in Section 2.1.2, each PI known to the system is associated with a distinctive set of parameters whose values control the operational characteristics of the PI. The specifications for each of the parameters associated with the PI whose internal identifier is PI<nn> are contained in a table called PPI<nn>. The attributes of PPI<nn> are the same as the attributes of PSE<nn> (see Section 2.4.2 and Table 2.2).

2.4.14 STORAGE OF PI VECTOR PARAMETER SPECIFICATIONS

The vector parameters of a PI are handled in the same way as those

ATTRIBUTE	TYPE	SIZE	DATA
SE_SEQ_NUM	INTEGER NUMERIC	4	SE usage number.
SIZE	INTEGER NUMERIC	2	Active size of clist.
C_NAME	CHARACTER	15	File name where the clist data is stored in the mainframe.
NUM_PTS	CHARACTER	15	Total number of data points.

Table 2.16

Attributes of the Relation SE<nn>C<ii>.

ATTRIBUTE	TYPE	SIZE	DATA
PI_NAME	CHARACTER	15	User specified name for the PI.
PI_CODE	CHARACTER	4	NEMS internal identifier for the PI.
CR_DATE	DATE	8	Creation date of the PI.
OWNER	CHARACTER	15	Name of the owner of the PI.
NUM_PAR	INTEGER NUMERIC	2	Number of parameters.
NUM_RPAR	INTEGER NUMERIC	2	Number of reflected parameters.
NUM_VAR	INTEGER MUMERIC	2	Number of output variables.
NUM_RVAR	INTEGER NUMERIC	2	Number of reflected output variables.
PRECISION	CHARACTER	1	Numerical precision (Single/Double).
NUM_USAGE	INTEGER NUMERIC	4	Total number of usages of the PI.
SE_MASTER	CHARACTER	15	Name of the SE to which the PI is bound.

Table 2.17

Attributes of the Relation PI_Lib.

of an SE (see Section 2.4.4). Suppose, in particular, that a PI whose internal identifier is PI<nn>, has a vector parameter whose PAR_CODE is P<ii>, then a supplementary relation PI<nn>I<ii> is created. PI<nn>I<ii> stores the default values, as well as upper and lower bounds for each component of the PI vector parameter of PI<nn>. The attributes of PI<nn>I<ii> are the same as those of SE<nn>I<ii> as given in Table 2.6.

2.4.15 STORAGE OF PI VARIABLE SPECIFICATIONS

Each PI known to the system is associated with a distinctive set of variables whose values capture a portion of the experiment results (see Section 2.1.2). The specifications for each of the variables associated with the PI whose internal identifier is PI<nn> are contained in a table called VPI<nn>. The attributes of VPI<nn> are the same as those of VSE<nn> as given in Table 2.7.

2.4.16 STORAGE OF ACTIVE VALUES FOR PI PARAMETERS

For each experiment initiated by NEMS, the parameters associated with the PI must be assigned active values which are used during the execution of the experiment. The active values for each of the

parameters associated with an PI whose internal identifier is PI<nn> are contained in a table called EPI<nn>. With the following two exceptions, the attributes of EPI<nn> are the same as those of ESE<nn> (see Table 2.13) :

(a) The attribute PI_SEQ_NUM replaces the attribute SE_SEQ_NUM of ESE<nn>. This attribute stores the usage number for the PI.

(b) The attribute SE_CODE replaces the attribute PI_CODE of ESE<nn>. This attribute stores the internal NEMS code of the SE to which the PI is bound.

2.4.17 STORAGE OF ACTIVE VALUES FOR PI VECTOR PARAMETERS

Suppose we have a PI whose internal identifier is PI<nn>, which has a vector parameter whose PAR_CODE is P<ii>. When NEMS initiates an experiment, the active values assigned to each of the vector components are maintained in a table called PI<nn>P<ii>. Except for the attribute SE_SEQ_NUM, the attributes of PI<nn>P<ii> are the same as those of SE<nn>P<ii> (see Table 2.14). The attribute SE_SEQ_NUM in SE<nn>P<ii> is replaced with PI_SEQ_NUM in PI<nn>I<ii>,where PI_SEQ_NUM is the usage number for the PI.

2.4.18 STORAGE OF VALUES FOR PI VARIABLES

As outlined in Section 2.1.2, each PI known to the system may be associated with a set of variables. These variables capture a portion of the results generated from the execution of an experiment with the PI. For a PI whose internal identifier is PI<nn>, the values of its variables are stored in the same table which stores the active values for the PI parameters; i.e., in the table EPI<nn>. The attributes of EPI<nn> are described in Section 2.4.16.

2.4.19 STORAGE OF VALUES FOR PI VECTOR VARIABLES

Suppose we have a PI whose internal identifier is PI<nn>, which has a vector variable whose VAR_CODE is V<ii>. When NEMS carries out an experiment, the value obtained for each of the vector components is placed in a table called PI<nn>V<ii>. With one exception, the attributes of PI<nn>V<ii> are the same as those of SE<nn>V<ii> (see Table 2.15). The exception is the attribute SE_SEQ_NUM in SE<nn>V<ii> which is replaced with PI_SEQ_NUM in PI<nn>V<ii>. PI_SEQ_NUM stores the usage number for the PI.

2.4.20 STORAGE OF VALUES FOR PI CLIST VARIABLES

Suppose we have a PI whose internal identifier is PI<nn>, which has a clist variable whose VAR_CODE is V<ii>. When NEMS carries out an experiment, the related information obtained for the clist variable is placed in a table called PI<nn>C<ii>. With one exception, the attributes of PI<nn>V<ii> are the same as those of SE<nn>V<ii> (see Table 2.16). The exception is the attribute SE_SEQ_NUM in SE<nn>C<ii> which is replaced with PI_SEQ_NUM in PI<nn>C<ii>. PI_SEQ_NUM stores the usage number for the PI.

2.4.21 STORAGE OF EXPERIMENTS

Each experiment initiated by NEMS is associated with a set of attributes whose values serve to characterize the experiment (see Section 2.1.3). This data is maintained in a table called EXP_Lib. The attributes of EXP_Lib are summarized in Table 2.18.

ATTRIBUTE	TYPE	SIZE	DATA
XID	INTEGER NUMERIC	4	Experiment ID.
SE_CODE	CHARACTER	4	NEMS internal identifier for the SE used in the experiment.
PI_CODE	CHARACTER	4	NEMS internal identifier for the PI used in the experiment.
EXPT_DATE	DATE	8	Experiment date.
SE_SEQ_NUM	INTEGER NUMERIC	4	SE usage number.
PI_SEQ_NUM	INTEGER NUMERIC	4	PI usage number.

Table 2.18

Attributes of the Relation EXP_Lib.

2.5 USER REQUIREMENTS

NEMS is a menu-driven system and it is very simple to use. After the user signs onto the system, he is guided by NEMS throughout the session. The user does not need to know anything about the internal structures or implementation details of NEMS in order to use NEMS effectively. However, the user must be familiar with the concepts of SE's and PI's, and their associated parameters and variables. Also, the user has to design the format of the input and output operations of the program code of each SE and PI according to prespecified conventions in order to ensure that NEMS can interface properly. Further discussion about user requirements is given in the following sections.

2.5.1 DEFINING THE SE/PI

The two basic entities of NEMS are the solution engine and the problem instance. As outlined in Section 2.1.1 and 2.1.2, each solution engine/problem instance is characterized by two groups of attributes; namely, the defining attributes and the operational attributes. At creation time, the user assigns a value to each of the defining attributes of an SE or a PI; these values remain invariant throughout the existence of the SE or the PI. The operational attributes of an SE are the active

values assigned by the user to the parameters for an experiment, and the values acquired by the variables after the execution of an experiment.

The defining attributes of an SE can be summarized as follows :

(1) SE_NAME which is the name of the SE;

(2) OWNER which is the name of the owner or creator of the SE;

(3) NUM_PAR which is number of parameters;

(4) NUM_RPAR which is the number of reflected parameters;

(5) NUM_VAR which is the number of variables;

(6) NUM_RVAR which is the number of reflected variables;

(7) PRECISION which is the numerical precision;

(8) VAR_NAME which is the name of a variable;

(9) VAR_TYPE which is the type of a variable;

(10) PAR_NAME which is the name of a parameter;

(11) PAR_TYPE which is the type of a parameter;

(12) N_TYPE which is the numeric type of a parameter or a variable;

(13) LENGTH which is the maximum length of a vector parameter or a vector variable;

(14) VIA which stores the name of an identifier through which the active length of a vector is obtained;

(15) LEN_CODE which is the active length code;

(16) DEFAULT which stores a default value of a parameter;

(17) LO_BOUND which stores a lower bound of a parameter;

(18) UP_BOUND which stores an upper bound of a parameter;

(19) EXEC_NAME which stores the name of the user-written exec file resident in the mainframe which invokes the program code of the SE and the PI.

Note that since the values of variables are acquired through the results produced by an experiment, the notion of default value, and upper and lower bounds does not apply to variables.

The set of defining attributes of a PI parallels that of an SE. The attribute SE_NAME is replaced by PI_NAME which is the name of the PI. In addition, the PI has an extra attribute SE_MASTER which stores the name of the SE to which the PI is bound. Also, EXEC_NAME is not an attribute of a PI.

2.5.2 INTERFACING WITH THE SE/PI

In order to execute an SE with a PI, i.e., in order to carry out an experiment, specific values must be assigned to the various parameters of both the SE and the PI. These values are acquired by NEMS at the PC

and must be transferred to the mainframe where the code for all SE's and PI's is assumed to be resident. After execution of an experiment, the generated results must be transferred back to the appropriate dBASE III storage structures on the PC.

The program codes of the SE and PI can access and create only text files. Therefore, in order to properly interface with these programs, all the required data for execution is first placed by NEMS into a text file which is then transferred to the mainframe. After the execution of an experiment, the experimental results are stored by the SE and PI code in a text file. This text file is transferred back to the PC where it is read by NEMS and the data values are placed into the appropriate dBASE III relations.

To ensure a proper interface between NEMS and the mainframe environment, all the input and output data relating to an experiment must be formatted according to a prespecified convention. Suppose NEMS activates an experiment with an SE called SE_NAME and a PI called PI_NAME. The records of the text files used for transferring data values must have the following features :

(a) Each data record contains a sequence of four values. These values, for example, can be the values of four scalar parameters, or the values

of four components of a vector parameter. Only the last data record is allowed to have less than four values.

(b) Each value in a data record is placed in a field with a field width of twenty, and each value must have a format which is consistent with its data type.

(c) In the case of parameter values transferred to the mainframe, the sequence of values in the data records corresponds to the sequence with which the parameters are defined during the creation activity. Furthermore, the parameters of the specified SE occur before those of the specified PI.

(d) For the case of a vector parameter, the value for each of the components is placed in the data records in natural sequence following the conventions of (a) and (b) above.

(e) After an experiment, the variable values produced by SE_NAME must be put into a data file called "SE_NAME OUTPUT", and the variable values produced by PI_NAME must be put into a data file called "PI_NAME OUTPUT". The sequence of values in the data records must correspond to the sequence with which the variables are defined during the creation activity.

(f) For the case of a clist variable, two data files must be created by the

corresponding SE or PI. The first must be called "SE_NAME CLIST" for the SE case, and "PI_NAME CLIST" for the PI case. The second must be called "SE_NAME NCLIST" for the SE case, and "PI_NAME NCLIST" for the PI case. The latter file, which contains the number of points in the clist, is appended as the last record to the corresponding output file of the SE or PI.

The actual values of the data points in the clist remain in the mainframe. Suppose the NEMS internal identifier of SE_NAME is SE<nn>, and the value of the SE_SEQ_NUM is <mmmm>; then the file of data point values (namely SE_NAME CLIST) is renamed as "SE<nn><mmmm> CLIST". Similarly, suppose the NEMS internal identifier of PI_NAME is PI<nn>, and the value of the PI_SEQ_NUM is <mmmm>; then the file of data point values (namely PI_NAME CLIST) is renamed by NEMS-remote (i.e. the NEMS EXEC file) as "PI<nn><mmmm> CLIST".

It is the user's responsibility to design the format of the input and output operations of the SE and PI program codes to conform with these conventions so that the data transfer can take place correctly.

The file transfer is explicitly initiated by the user and the user must be aware of some basic aspects of the file transfer procedure. To simplify the procedure, NEMS puts all the information required for the

experiment into a single file called "NEMS.TXT". "NEMS.TXT" consists of three parts. The first part contains CMS commands, the second part contains the SE parameter active values and the third part contains the PI parameter active values.

When the user is signed onto the mainframe, he must explicitly initiate NEMS-remote which is an exec file called NEMS EXEC that is resident in the mainframe. This is achieved by issuing the command "NEMS". NEMS-remote breaks the file "NEMS TXT" into three separate files : "SE EXEC" which contains CMS commands ; "SE_NAME INPUT" which contains the SE parameter active values and serves as the input file to the program code of SE_NAME; and "PI_NAME INPUT" which contains the PI parameter active values and serves as the input file to the program code of PI_NAME. NEMS-remote then initiates SE EXEC which in turn activates a mainframe resident file which executes the SE/PI combination. A representation of this activity is given in Figure 2.6.

At the end of the experiment, NEMS-remote organizes (via its SE EXEC component) all the output data produced by the experiment. In the most general case, SE_NAME produces three files called "SE_NAME OUTPUT", "SE_NAME CLIST" and "SE_NAME NCLIST"; and PI_NAME

similarly produces three files called "PI_NAME OUTPUT", "PI_NAME CLIST" and "PI_NAME NCLIST". As discussed in (e) and (f), the values of the total number of data points of any clists (as contained in SE_NAME NCLIST and PI_NAME NCLIST) are appended to the corresponding output files, and the files which store the actual values of the clist data points are renamed by NEMS-remote according to specific conventions. Then NEMS-remote appends the file "PI_NAME OUTPUT" to the end of the file "SE_NAME OUTPUT", and renames the resulting file as "NEMS OUTPUT". The user then must initiate another file transfer to transfer these experimental results back to the PC. NEMS reads the data into "NEMS.OUT" and places the data values into the appropriate NEMS relations. A representation of this activity is given in Figure 2.7.

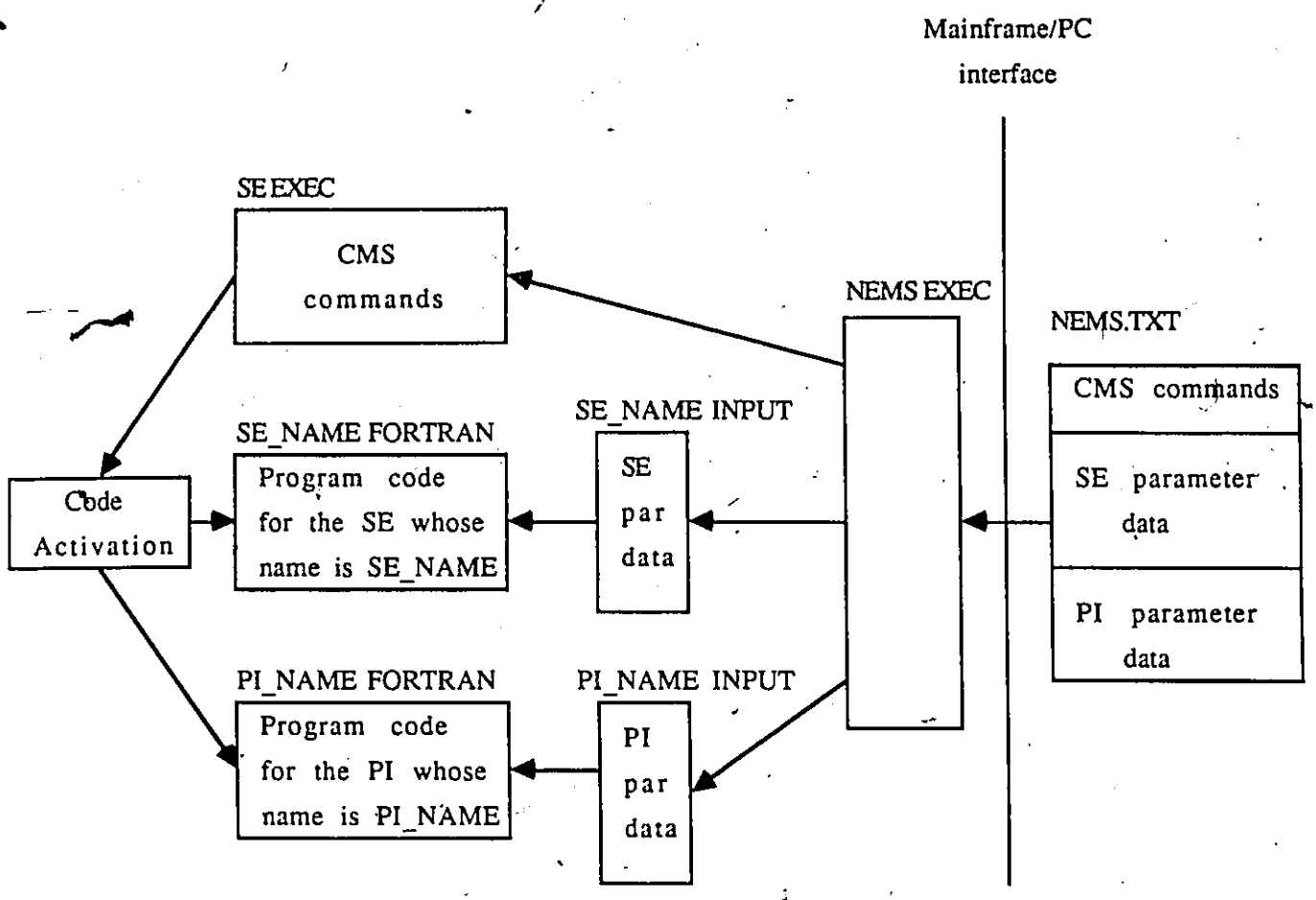
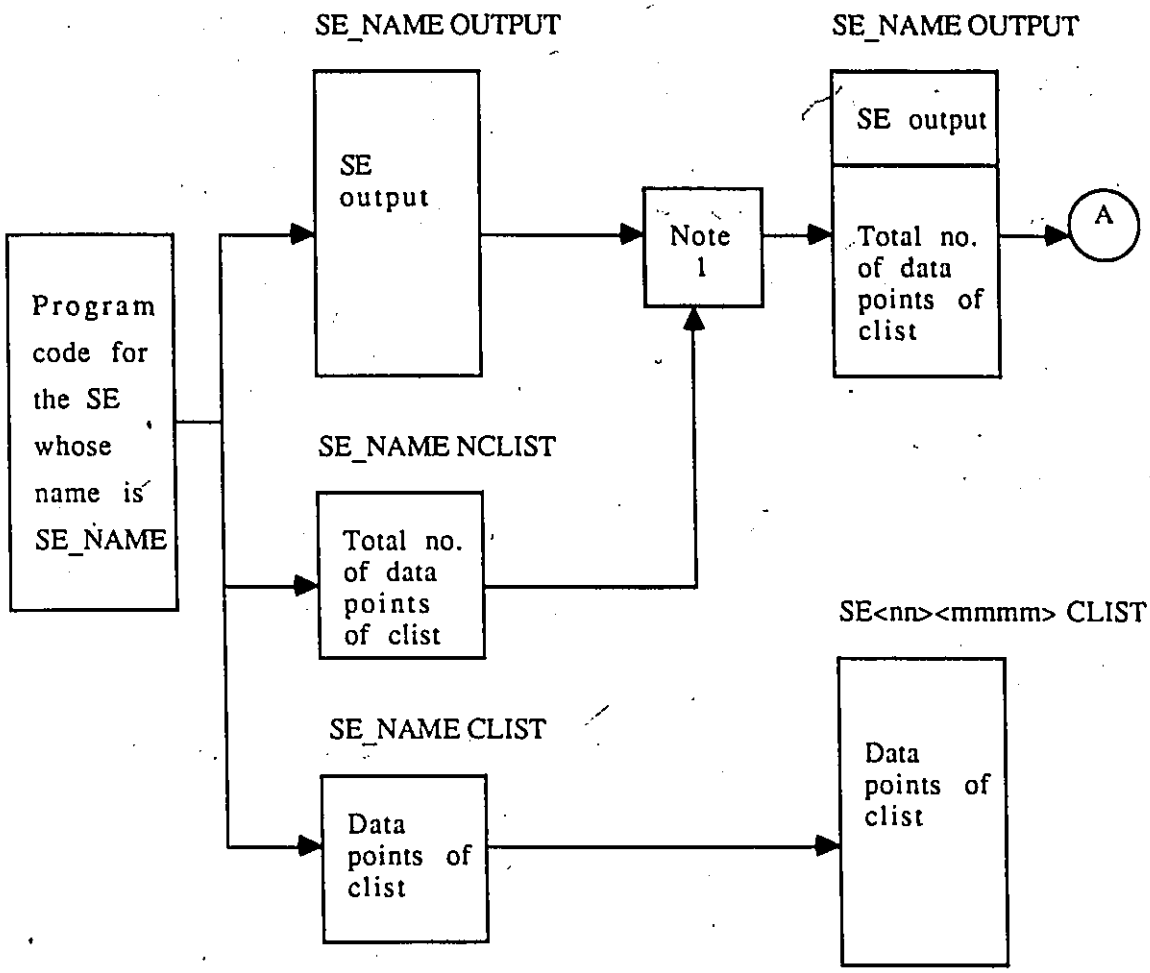


Figure 2.6

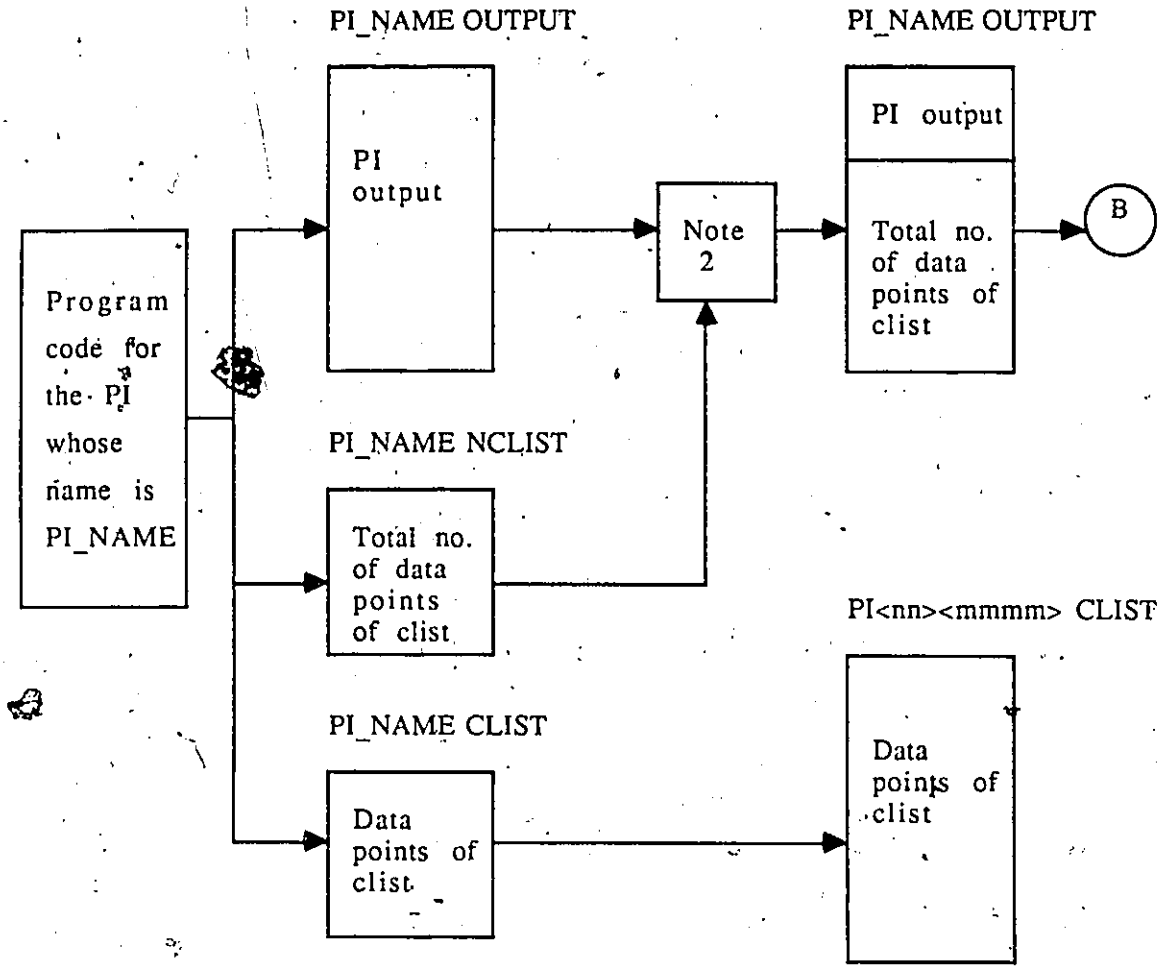
Representation of the PC to Mainframe Transfer Activity.



Note 1 : Append the value of the total no. of data points of clist as the last record of SE_NAME OUTPUT.

Figure 2.7

Representation of the Mainframe to PC Transfer Activity.



Note 2 : Append the value of the total no. of data points of clist as the last record of PI_NAME OUTPUT.

Figure 2.7 (Cont'd)

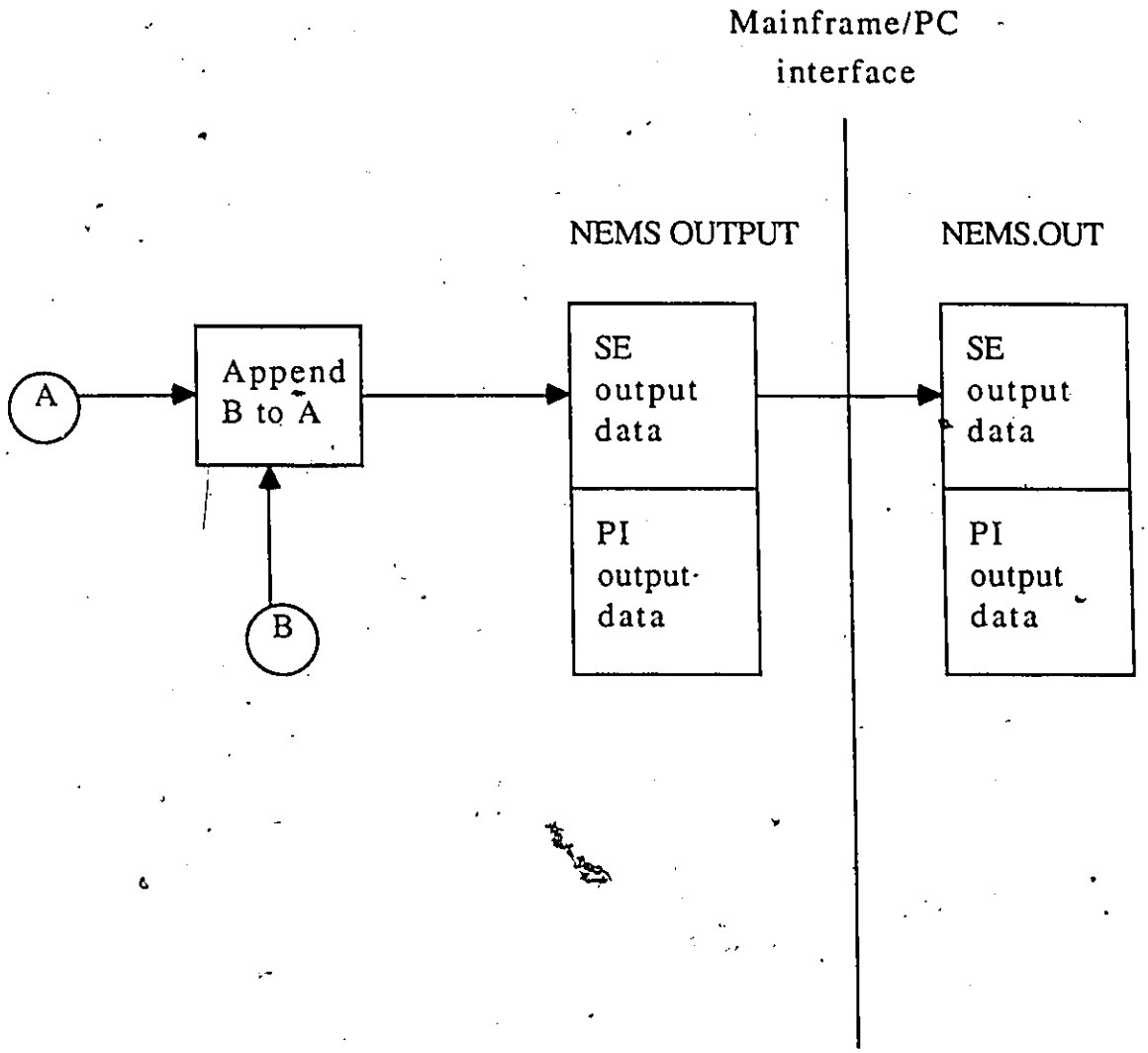


Figure 2.7 (Cont'd)

CHAPTER 3

IMPLEMENTATION ASPECTS OF THE NEMS MODES

3.1 THE CREATE MODE

As pointed out in Section 2.3.1, the user creates the existence of SE's and PI's in the create mode. When an SE or PI is created in NEMS, the values of its defining attributes are stored in the SE_Lib or the PI_Lib relation as appropriate. Figure 2.1 summarizes the basic steps of the create mode. A discussion of the create mode is given in the following sections.

3.1.1 NAMING CONVENTION FOR RELATIONS IN NEMS

The relations in NEMS are maintained as named dBASE III tables. NEMS assigns an internal identifier to each SE or PI that is created. This identifier is in the form of SE<nn> for solution engines, and PI<nn> for problem instances, where <nn> represents a two digit identifier. Suppose we have a solution engine whose internal identifier is SE<nn>. Each of the relations associated with this SE is assigned a name such that part of its name is SE<nn>. For example, the relation which maintains the SE parameter specifications is assigned the name PSE<nn>.

Suppose we have an SE vector parameter whose PAR_CODE is P<ii>. The relation SE<nn>P<ii> stores the active values for each component of P<ii>, and the relation SE<nn>I<ii> stores information, such as the default value and permissible range for each component of P<ii>.

By constructing all the NEMS relations according to this naming convention, the purpose of any given relation can be easily identified just by examining its name. Also, when NEMS performs data entry or data retrieval, the name of the relation required for the operation can be obtained easily, and NEMS can rapidly access the appropriate relation. In the program codes of the NEMS software, the names of the various relations are obtained by concatenating a simple code, such as a single letter, with the internal identifier of an SE or a PI as appropriate.

Suppose an SE whose internal identifier is SE<nn> has been created in NEMS, and it has a vector parameter whose PAR_CODE is P<ii>, a vector variable whose VAR_CODE is V<jj> and a clist variable whose VAR_CODE is V<kk>. To obtain the name of the relation which stores the specifications of the parameters, the software concatenates the code "P" with the internal identifier of the SE, and obtains the name "PSE<nn>". To obtain the name of the relation which stores the specifications of the reflected parameters, the software concatenates the code "RP" with the

internal identifier of the SE, and obtains the name "RPSE<nn>". Similarly, the various other relations that are associated with SE<nn> are named in a convenient and relatively suggestive way. Given below is a list of these relations and their respective functional purpose (in the interest of completeness, the list includes PSE<nn> and RPSE<nn> which are described above) :

(a) PSE<nn> stores the specifications of the parameters;

(b) RPSE<nn> stores the specifications of the reflected parameters;

(c) VSE<nn> stores the specifications of the variables;

(d) RVSE<nn> stores the specifications of the reflected variables;

(e) SE<nn>I<ii> stores the specifications of a vector parameter;

(f) ESE<nn> stores the active values assigned to the scalar parameters, and the values acquired by the scalar variables;

(g) SE<nn>P<ii> stores the active values assigned to each component of the vector parameter whose internal code is P<ii>;

(h) SE<nn>V<jj> stores the values acquired by each component of the vector variable whose internal code is V<jj>;

(i) SE<nn>C<kk> stores the data relating to the clist variable whose internal code is C<kk>.

An entirely analogous convention is used to name all the relations

that are associated with a problem instance having an internal identifier of PI<nn>. A list of these is given below :

(a) PPI<nn> stores the specifications of the parameters;

(b) VPI<nn> stores the specifications of the variables;

(c) PI<nn>I<ii> stores the specifications of a vector parameter;

(d) EPI<nn> stores the active values assigned to the scalar parameters, and the values acquired by the scalar variables;

(e) PI<nn>P<ii> stores the active values assigned to each component of the vector parameter whose internal code is P<ii>;

(f) PI<nn>V<jj> stores the values acquired by each component of the vector variable whose internal code is V<jj>;

(g) PI<nn>C<kk> stores the data relating to the clist variable whose internal code is C<kk>.

3.1.2 BASIC STRUCTURES ASSOCIATED WITH THE CREATE MODE

The structure of the relations associated with each SE and with each PI in NEMS is basically the same. Variations arise due to the different numbers of parameters and variables (in ESE<nn> and EPI<nn>, see Sections 2.4.7 and 2.4.16), and the different vector lengths (in

SE<nn>I<ii>, SE<nn>P<ii>, SE<nn>V<jj>, and PI<nn>I<ii>, PI<nn>P<ii>, PI<nn>V<jj>, see Sections 2.4.4, 2.4.8, 2.4.10, 2.4.14, 2.4.17 and 2.4.19). Clearly, it would be desirable to have these new database relations created automatically. However, dBASE III is designed to create database structures in an interactive mode, and thus this creation process cannot be automated.

The dBASE III template features provide the means for dealing with this situation. After the creation of a database template, copies of it can be reproduced, and dBASE III allows the reproduced copies to contain all or a subset of the attributes defined to the original, but the name, type and field width of each of the attributes of the copy must be the same as that of the original. Consequently, to deal with the problem of automating the creation process, NEMS maintains a collection of basic database structures and these serve as the templates for the creation of new relations. Each time a new SE or PI is created, NEMS selects the appropriate templates for the necessary relations, and uses information about the number of parameters and variables, and also the maximum lengths of all its vectors to make reproductions of the templates. These reproduced copies are assigned appropriate names according to the internal coding conventions. The procedure ensures that they contain

the appropriate number of fields for maintenance of parameter and variable information, and the appropriate number of components for each of the vector parameters/variables.

Given below is a brief summary of these template relations :

(a) St_Par is used as the template for the relations PSE<nn> and PPI<nn>;

(b) St_RPar is used as the template for the relation RPSE<nn>;

(c) St_Vect is used as the template for the relations SE<nn>I<ii> and PI<nn>I<ii>;

(d) St_Var is used as the template for the relations VSE<nn>, RVSE<nn> and VPI<nn>;

(e) St_VSE is used as the template for the relations SE<nn>P<ii> and SE<nn>V<jj>;

(f) St_VPI is used as the template for the relations PI<nn>P<ii> and PI<nn>V<jj>;

(g) St_ESE is used as the template for the relation ESE<nn>;

(h) St_EPI is used as the template for the relation EPI<nn>;

(i) St_SEC is used as the template for the relation SE<nn>C<kk>;

(j) St_PIC is used as the template for the relation PI<nn>C<kk>.

3.1.3 SPECIFICATIONS OF PARAMETERS AND VARIABLES

With each SE and PI created in NEMS, there is normally an associated set of parameters. Each of these parameters can be of type scalar numeric, vector numeric or character string. At creation time, one of the user requirements is to specify the name and the type of each parameter. The user may also assign a default value to each parameter (once specified, this value remains invariant). If the parameter is of type vector numeric, the maximum length and relevant information about its active length are also obtained from the user. For an SE whose internal identifier is SE<nn>, all of the above parameter specifications are stored in a table called PSE<nn>. For a PI whose internal identifier is PI<nn>, all of the above parameter specifications are stored in a table called PPI<nn>.

For the case of a vector parameter, a default value may be assigned to each of the vector components. Suppose an SE, whose internal identifier is SE<nn>, has a vector parameter whose PAR_CODE is P<ii>, then the default value of each of its vector components is stored in a table called SE<nn>I<ii>. Suppose a PI, whose internal identifier is PI<nn>, has a vector parameter whose PAR_CODE is P<ii>, then the default value of each of its vector components is stored in a table called

PI<nn>I<ii>.

At execution time, if the user does not assign a specific active value to a parameter, the default value is used as the active value for execution.

The user may also assign a permissible range to a parameter or to a component of a vector parameter when it has a numeric type. An active value assigned by the user at execution time is checked against this range, and a violation is brought to the user's attention. An assigned default value for a parameter must, of necessity, fall within the range specification.

The specifications for the variables of an SE whose internal identifier is SE<nn> are stored in a table called VSE<nn>. The specifications for the variables of a PI whose internal identifier is PI<nn> are stored in a table called VPI<nn>. The various specifications for a variable of an SE/PI are the same as those of a parameter of an SE/PI, except that the concepts of a default value and a permissible range do not apply to a variable.

3.1.4 SPECIFICATIONS OF SE REFLECTED PARAMETERS AND VARIABLES

As noted in Section 2.1.1, a solution engine can have associated with it two special groups of identifiers called "reflected parameters" and "reflected variables" respectively. These special groups of parameters and variables have relevance to those situations where every problem instance associated with the solution engine includes a common set of parameters and/or variables. The "reflected" concept makes it possible to specify such a common set of identifiers only once at the time when the solution engine itself is being created. Every subsequent definition of a problem instance that is associated with the solution engine then inherits the reflected parameters and/or variables that are so defined. Thus, although such parameters/variables are specified at the time of solution engine creation, their existence is manifested in the context of subsequently defined problem instances. This, in particular, avoids the need to redefine such parameters/variables for each new problem instance.

Suppose we have an SE whose internal identifier is SE<nn>. The specifications for the set of SE reflected parameters (if such exist) are stored in a table called RPSE<nn> (see Section 2.4.3). The specifications

for the set of SE reflected variables are stored in a table called RVSE<nn>. Each of these parameters can be of type scalar numeric, vector numeric or character. If the parameter is of type vector numeric, the maximum length and relevant information about its active length are also obtained during the dialog relating to the solution engine creation. Note that the user does not assign a default value or a permissible range to reflected parameters. This information is entered during the creation of each specific problem instance that is associated with the solution engine.

Similar information is obtained for each SE reflected variable. If the variable is of type vector numeric, the maximum length and relevant information about its active length are obtained during the dialog relating to the solution engine creation. If the variable is of type clist, the maximum size and relevant information about its active size are also obtained during the creation dialog.

To illustrate the inheritance mechanism associated with the reflected identifier concept, suppose SE<nn> is a solution engine that has a set of reflected parameters. Suppose furthermore that a problem instance, which has SE<nn> as its master, is to be created. During the creation procedure NEMS automatically moves the reflected parameters

into the PPI<nn> table (see Section 2.4.16) that is linked to the PI; i.e. the available specification information for the reflected parameters is entered into the appropriate tuple positions in PPI<nn>. The user (i.e. PI creator) is then prompted for specialized values such as default and range values for each of the parameters that has been automatically created in this way. NEMS then provides the opportunity for the entry of additional parameter specifications that may be distinctive to the specific PI being constructed. The handling of reflected variables takes place in an analogous way.

3.1.5 PROGRAM STRUCTURE FOR THE CREATE MODE

The program module for the create mode contains two main routines, namely, Cr_SE and Cr_PI which are used respectively for the creation of solution engine and the creation of problem instances. They are called by the routine Cr. Figure 3.1 gives a representation of the program structure which relates to the create mode.

The routine Cr_SE creates an entry in the relation SE_Lib and creates, as necessary, the relations PSE<nn>, RPSE<nn>, VSE<nn>, RVSE<nn>, ESE<nn> together with their supplementary relations such as SE<nn>I<ii>, SE<nn>P<ii>, SE<nn>V<ii>, SE<nn>C<ii>. Six different routines

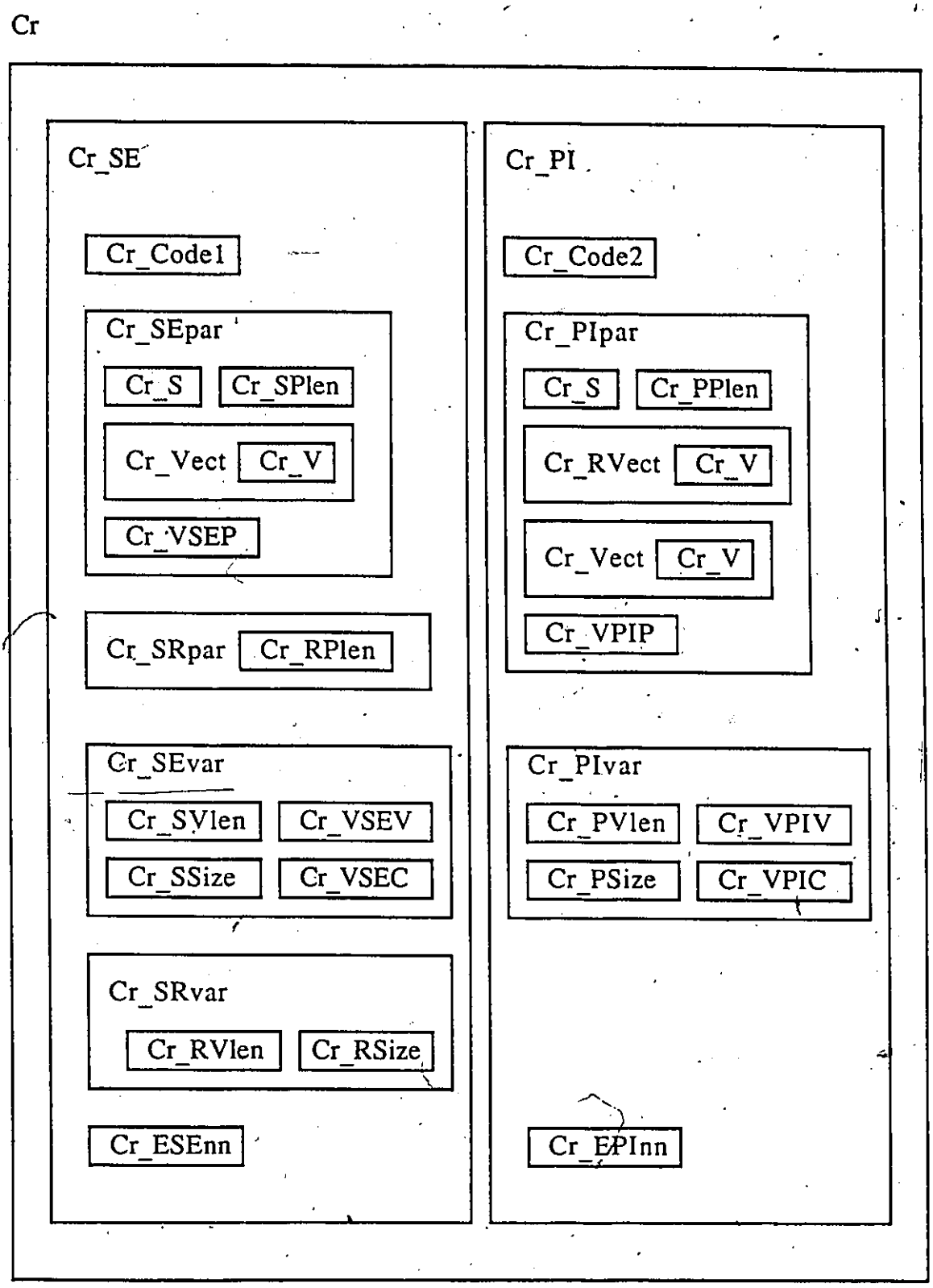


Figure 3.1

Program Structure for the Create Mode.

are, in turn, called by Cr_SE to carry out its various tasks; these are summarized below :

(a) Cr_Code1 : generates the NEMS internal identifier for the SE;

(b) Cr_SEpar : creates the PSE<nn> table which stores the specifications for the SE parameters. Cr_SEpar, in turn calls the routines Cr_S to obtain the default value of a scalar parameter, Cr_SPlen to obtain the specifications for the active length of a vector parameter, Cr_Vect to create the relation SE<nn>I<ii> (this actually takes place via a call to the routine Cr_V) and obtain the upper and lower bounds, as well as the default value of each component of a vector parameter, and Cr_VSEP to create the relation SE<nn>P<ii> which stores the active values of a vector parameter.

(c) Cr_SRpar : creates the RPSE<nn> relation which stores the specifications for the SE reflected parameters. Cr_SRpar in turn, calls the routine Cr_RPlen to obtain specifications for the active length of an SE reflected vector parameter.

(d) Cr_SEvar : creates the VSE<nn> relation which stores specifications for the SE variables. Cr_SEvar, in turn calls the routines Cr_SVlen to obtain the active length specifications for a vector variable, Cr_VSEV to create the table SE<nn>V<ii> which stores experimental results acquired

by a vector variable, Cr_SSize to obtain the active size specifications of a clist variable, and Cr_VSEC to create the table SE<nn>C<ii> which stores information relating to a clist variable after the execution of an experiment.

(e) Cr_SRvar : creates the RVSE<nn> table which stores the specifications for the SE reflected variables. Cr_SRvar in turn, calls the routines Cr_RVlen to obtain specifications for the active length of an SE reflected vector variable, and Cr_RSize to obtain specifications for the active size of an SE reflected clist variable.

(f) Cr_ESEnn : creates the ESE<nn> table which stores the active values of parameters and the experimental results acquired by scalar variables.

The routine Cr_PI creates an entry in the relation PI_Lib and creates, as necessary, the relations PPI<nn>, VPI<nn>, EPI<nn> together with their supplementary relations such as PI<nn>I<ii>, PI<nn>P<ii>, PI<nn>V<ii>, PI<nn>C<ii>. To carry out its various functions, Cr_PI calls four different routines which are summarized below :

(a) Cr_Code2 : generates the NEMS internal identifier for the PI;

(b) Cr_PIpar : creates the PPI<nn> table which stores the specifications for the PI parameters. Cr_PIpar, in turn calls the routines Cr_S to

obtain the default values for scalar parameters, Cr_PPlen to obtain the specifications for the active length of a vector parameter, Cr_Vect (or Cr_RVect for the case of a reflected vector parameter) to create the relation PI<nn>I<ii> (this actually takes place via a call to the routine Cr_V) and obtain the upper and lower bounds, as well as the default value of each component of a vector parameter, and Cr_VPIP to create the relation which stores the active values of a vector parameter.

(c) Cr_PIvar : creates the VPI<nn> table which stores the specifications for the PI variables. Cr_PIvar, in turn calls the routines Cr_PVlen to obtain the active length specifications for a vector variable, Cr_VPIV to create the table PI<nn>V<ii> which stores experimental results acquired by a vector variable, Cr_PSize to obtain the active size specifications for a clist variable, and Cr_VPIC to create the table PI<nn>C<ii> which stores information relating to a clist variable after the execution of an experiment.

(d) Cr_EPInn : creates the EPI<nn> table which stores the active values of parameters and the experimental results acquired by scalar variables.

3.2 THE EXECUTE MODE

In the execute mode, NEMS carries out an experiment and creates an instance of an experiment object. The basic steps in the execute mode are given in Figure 2.2. The following sections discuss some implementational aspects of the execute mode.

3.2.1 AN INSTANCE OF AN EXPERIMENT OBJECT

A basic action which takes place when an experiment is to be executed under NEMS, is the creation of an instance of an experiment object. This instance is stored as a tuple in a relation called EXP_Lib (see Section 2.4.21). This tuple contains 6 attributes, namely, XID, SE_CODE, PI_CODE, EXPT_DATE, SE_SEQ_NUM and PI_SEQ_NUM. The XID value is a sequential numeric identifier for the experiment. The SE_CODE and PI_CODE are the internal identifiers for the SE and PI that are to be used in the experiment. EXPT_DATE is the current date. The SE_SEQ_NUM is a sequential usage number for the SE in question. Similarly, the PI_SEQ_NUM is a sequential usage number for the PI in question.

Suppose the internal identifier of the SE used in an experiment is SE<nn>, and that of the PI is PI<nn>. The execution of an experiment also results in the creation of a tuple in the ESE<nn> relation as described in

Section 2.4.7. This table has an attribute SE_SEQ_NUM (its key) and the value stored is the same as that stored in EXP_Lib. Similarly, the PI_CODE attribute of this table is assigned the same value that was stored in EXP_Lib. In the event that the SE has vector parameters or vector variables or a clist variable, then the SE_SEQ_NUM value is also stored in the appropriate location of the supplementary relations SE<nn>P<ii>, SE<nn>V<ii> and SE<nn>C<ii>.

In a similar way, the execution of an experiment also results in the creation of a tuple in the relation EPI<nn> which is uniquely identified by the PI_SEQ_NUM value stored in EXP_Lib. The SE_CODE value used in EXP_Lib is also stored in EPI<nn>. The treatment of vector parameters, vector variables and a clist is analogous to that described above.

The assignments described above can be viewed as an initialization procedure for the experiment. Before the experiment can actually be carried out, active values for the relevant parameters must be established. This is outlined below.

3.2.2 ASSIGNMENT OF ACTIVE VALUES TO PARAMETERS

The initiation of an experiment requires that the user first identify the SE and the PI to be used in the experiment and the internal

identifiers for the SE and PI are then both stored in the table EXP_Lib. Then active values must be assigned to each of the SE and PI parameters. Suppose the internal identifiers for the SE and the PI are SE<nn> and PI<nn> respectively. Using information from the PSE<nn> and PPI<nn> tables, NEMS displays the name, the type and the default value of each of the parameters. The user is then expected to enter the active values for each of the parameters to be used in the experiment. Each value assigned by the user is checked for consistency with the prescribed type and range values. The assigned values are stored in the ESE<nn> and EPI<nn> tables as appropriate. If the user does not assign a value to a parameter, the default value is assigned as the active value for the parameter.

After acquiring an active value for each of the scalar parameters, NEMS proceeds to obtain active values for the components of each vector parameter. However, before doing so, the active length of the vector parameter in question must first be established. Recall that various options for the establishment of active length are possible and at creation time a specific option is associated with each vector parameter. This option is invoked in the execute mode and may or may not give rise to the need for explicit user input.

If the active length of an SE vector parameter has been specified as

"fixed", then the active length is simply the maximum length of the vector. If the active length is specified as "user-assignable at execution time", then the user will be prompted to explicitly provide the active length of the vector parameter. If the active length is specified as "via the value of a scalar parameter of the SE", then NEMS determines the identifier stated in the corresponding VIA attribute in the relation PSE<nn>. NEMS then assigns the value of this SE scalar parameter as the active length of the vector parameter in question (notice that this identifier appears in the VIA attribute will already have been assigned a value at this point). If the active length is specified as "via the active length of another SE vector parameter", then NEMS again determines the identifier stated in the corresponding VIA attribute in the relation PSE<nn>. If, for example, this identifier is called XYZ, then NEMS will assign the active length of the vector parameter in question to be the same as the active length of XYZ (notice that the active length of XYZ will already have been established at this point because of sequencing checks that are carried out in the create stage).

The active length of an SE vector parameter can also be specified as "via the value of a scalar parameter of the PI", or "via the active length of a vector parameter of the PI", or "via the active length of a vector

variable of the PI". This cross referencing mechanism is discussed in Section 3.2.4.

After establishing the active length of a vector parameter, NEMS prompts the user for a value for each of its active components. NEMS begins by displaying the name, the type, the maximum size and the active size of the vector parameter. The component number for each active component is then displayed together with the associated default value. If the user does not assign a specific value to an active component, then the default value is assigned.

The active length of the vector parameter and the values of its (active) components are stored in the relation SE<nn>P<ii>, where P<ii> is the PAR_CODE of the vector parameter in question. The name of this table is stored in the attribute P<ii> of the ESE<nn> table.

The steps involved in handling vector parameters of a PI are analogous to those outlined above for an SE.

3.2.3 ACQUISITION OF INFORMATION ABOUT VARIABLES

As outlined in Sections 2.1.1 and 2.1.2, the variables of an SE and a PI serve to capture experimental results. Before initiating an experiment, the user may be required to provide information about some of the

vector variables and/or clist variables of the SE/PI. This depends on the option that has been specified for the establishment of the active length of the vector variables and the active size of the clist variables at creation time. This information is stored in the LEN_CODE and VIA attributes of VSE<nn> and VPI<nn>.

If the active length of an SE vector variable is specified as "fixed", or "user-assignable at execution time", or "via the value of a scalar parameter of the SE", or "via the active length of another vector parameter of the SE", then the active length of this vector variable is obtained in the same way as the active length of an SE vector parameter as described in Section 3.2.2.

The case where the active length is specified as "via the active length of another vector variable of the SE" is handled in an analogous way to the case where the active length of an SE vector parameter is specified to be "via the active length of another vector parameter of the SE".

NEMS must also establish the active size of a clist variable at execution time. If the active size of an SE clist variable is specified as "fixed", then the active size is simply the maximum size of the clist. If the active size is specified as "user-assignable at execution time", then the user will be prompted for an input which gives the active size of the clist

variable. If the active size is specified as "via the value of a scalar parameter of the SE", then NEMS determines the identifier stated in the corresponding VIA attribute of the clist variable and then assigns the value of this SE scalar parameter to be the active size of the clist variable in question (notice that the identifier which appears in the VIA attribute will have already been assigned a value at this point).

The active length of an SE vector variable can also be specified as "via the value of a scalar parameter of the PI", or "via the active length of a vector parameter of the PI", or "via the active length of a vector variable of the PI". Furthermore, the active size of an SE clist variable can be specified as "via the value of a scalar parameter of the PI". Such interdependence between an SE and a PI is discussed further in Section 3.2.4.

The handling of PI vector and clist variables is analogous to that outlined above for an SE.

3.2.4 CROSS REFERENCE BETWEEN THE SE AND PI

The need for cross referencing can occur under several circumstances. These are when the active length of an SE vector depends on the value of a PI scalar parameter, or the active length of a PI vector

parameter, or the active length of a PI vector variable. Or alternately, it can occur when the active size of an SE clist variable depends on the value of a PI scalar parameter, or the active size of a PI clist variable. Similarly, when the active length of a PI vector or the active size of a PI clist depends on some values from the SE, then there again is a need for cross referencing.

It should be stressed that cross referencing is permitted in NEMS in only one direction; i.e. if a length/size value in an SE depends on the PI, then it is forbidden to simultaneously have a length/size value in the PI depending on the SE. Correspondingly, when an experiment is being initiated in the execute mode, NEMS takes into account cross referencing requirements to establish whether SE active values should be obtained from the user before PI active values, or vice versa.

Suppose that the active length of an SE vector depends on the PI. This implies that some PI identifier appears in the VIA attribute of the SE vector. NEMS determines this identifier among the PI parameters and variables, i.e. in the PPI<nn> and VPI<nn> tables. Suppose this identifier is called XYZ. If XYZ is a PI scalar parameter, then NEMS assigns the value of XYZ to be the active length of the SE vector in question. This value is obtained from the EPI<nn> table. If, on the other hand, XYZ is a PI vector

parameter or variable, then NEMS assigns the active length of the vector parameter in question to be the same as the active length of XYZ. This information is obtained either from PI<nn>P<ii> (if XYZ is a vector parameter), or from PI<nn>V<ii> (if XYZ is a vector variable).

Similarly, if the active size of an SE clist cross references a PI value, then NEMS searches among the PI parameters and variables for the PI identifier stated in the corresponding VIA attribute of the clist. Suppose this identifier is called XYZ. If XYZ is a PI scalar parameter, then NEMS assigns the value of XYZ to be the active size of the SE clist in question. However, if XYZ is a PI clist variable, then NEMS assigns the active size of the clist in question to be the same as the active size of XYZ.

3.2.5 PROGRAM STRUCTURE FOR THE EXECUTE MODE

The programs in the execute module are invoked during the execution of an experiment. A representation of the program structure of the execute module is given in Figure 3.2. The various programs which comprise this module, are outlined below.

(a) Ex : carries out the execution of an experiment. It calls the routines which are listed in (b), (c), (d), (e) and (f) below.

(b) Ex_1 : obtains the active values for SE and PI parameters, and the

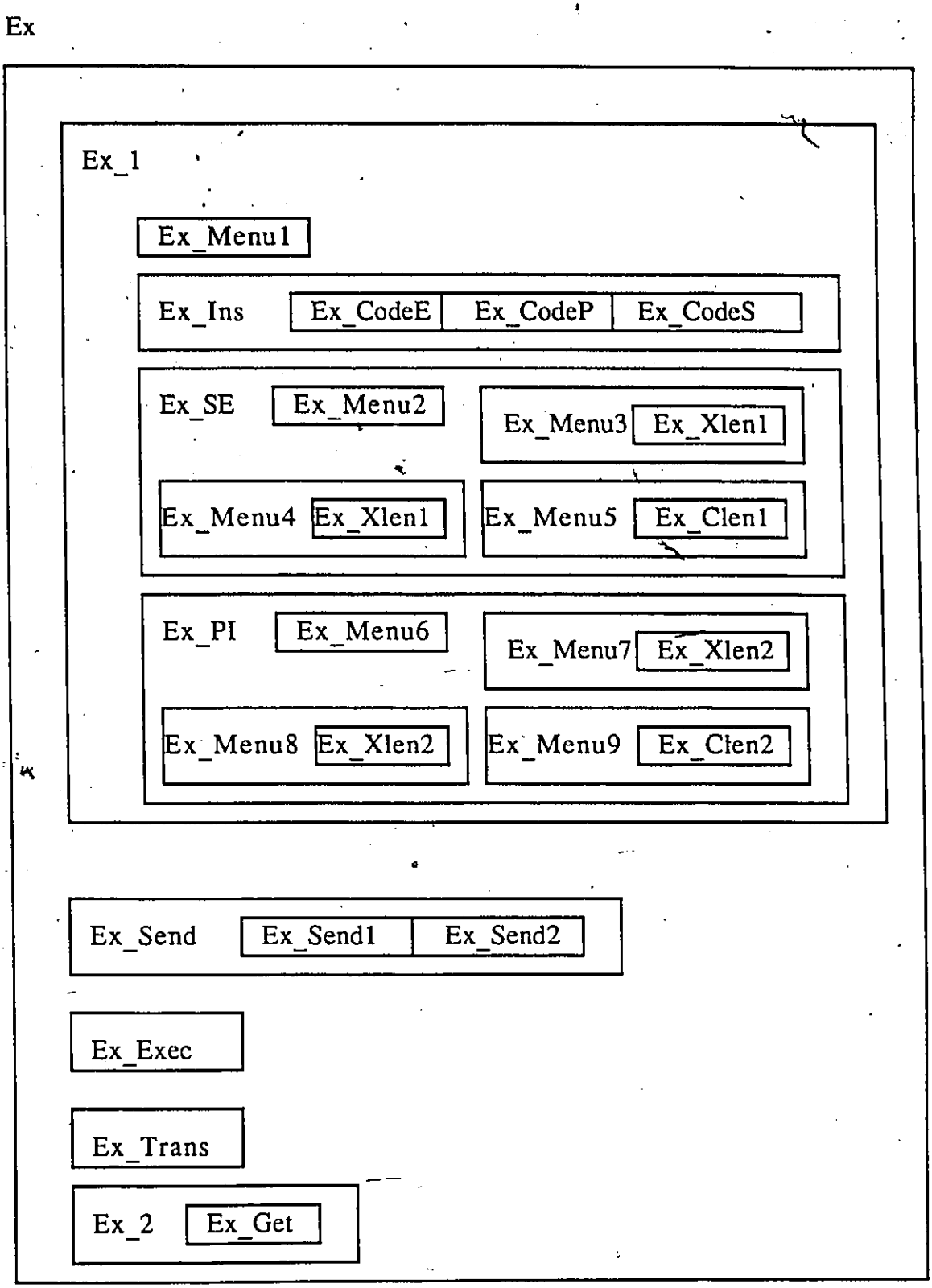


Figure 3.2

Program Structure for the Execute Mode.

active length/size of SE and PI variables. Ex_1 in turn, calls four routines, namely, Ex_Menu1, Ex_Ins, Ex_SE and Ex_PI. Ex_Menu1 prompts the user for the names of the SE and PI to be used in the experiment. Ex_Ins generates the appropriate XID, SE_seq_num and PI_seq_num for the experiment, and inserts this information in the relations EXP_Lib, ESE<nn>, SE<nn>P<ii>, SE<nn>V<ii>, SE<nn>C<ii>, EPI<nn>, PI<nn>P<ii>, PI<nn>V<ii> and PI<nn>C<ii> as appropriate. Ex_SE calls the routines Ex_Menu2 to obtain the active values of SE scalar parameters, Ex_Menu3 to obtain the active values for the components of each SE vector parameter, Ex_Menu4 to obtain the active length of each SE vector variable, and Ex_Menu5 to obtain the active size of an SE clist variable. Note that if the active length of an SE vector depends on a value in the PI, then Ex_Menu3 and Ex_Menu4 each calls the routine Ex_Xlen1 which searches the PPI<nn> and VPI<nn> relations for the necessary active length information. Similarly, if the active size of an SE clist depends on a value in the PI, then Ex_Menu5 calls the routine Ex_Clen1 which searches the PPI<nn> and VPI<nn> relations for the necessary active length information. In a similar way, Ex_PI calls the routines Ex_Menu6 to obtain the active values of PI scalar parameters, Ex_Menu7 to obtain the active values for the components of each PI vector parameter, Ex_Menu8 to obtain the

active length of a PI vector variable, and Ex_Menu9 to obtain the active size of a PI clist variable. Note that if the active length of a PI vector depends on a value in an SE, then Ex_Menu7 and Ex_Menu8 each calls the routine Ex_Xlen2 to obtain the necessary active length information. Also, if the active size of a PI clist depends on a value in an SE, then Ex_Menu9 calls the routine Ex_Clen2 to obtain the active size information.

(c) Ex_Send : organizes the active parameter values of the SE and PI as records in a file. This is achieved by calls to Ex_Send1 and Ex_Send2 which have the task of formatting the SE parameter values and PI parameter values respectively.

(d) Ex_Exec : creates the file NEMS.TXT which is processed in the mainframe. This file consists of three parts; namely, CMS commands, the active values of the SE parameters and the active values of the PI parameters.

(e) Ex_Trans : assists the user in performing the file transfer;

(f) Ex_2 : places the experimental results into the appropriate tuples in the relations ESE<nn>, SE<nn>V<ii>, SE<nn>C<ii>, EPI<nn>, PI<nn>V<ii> and PI<nn>C<ii> as appropriate. Ex_2 calls the routine Ex_Get which extracts values from the output file NEMS.OUT that is created at the mainframe, and transferred to the PC.

3.3 THE QUERY MODE

As outlined in Section 2.3.3, it is in the query mode that a user is able to browse through the data that has been accumulated in the NEMS database. The basic steps of the query mode are given in Figure 2.3. Some implementational aspects of the query mode are described in the following sections.

3.3.1 DESCRIPTION OF THE QUERY SUBSYSTEM

Large amounts of data will typically be accumulated in the NEMS database. Since the NEMS system is built upon dBASE III, the user naturally has the option of accessing the data tables directly through the retrieval facilities provided by this underlying system. However, this route will generally not be attractive for most users because it requires that the user be very familiar with not only the internal structure of the data tables, but also the dBASE III retrieval system.

As an alternative, NEMS includes a menu driven query subsystem which provides a simplified means for browsing through the NEMS database. This query subsystem provides three basic menus for the user, namely, the SHOW/DESIGNATE menu, the REFINE menu and the DISPLAY menu. The structure of this query subsystem is shown in Figure 3.3. The

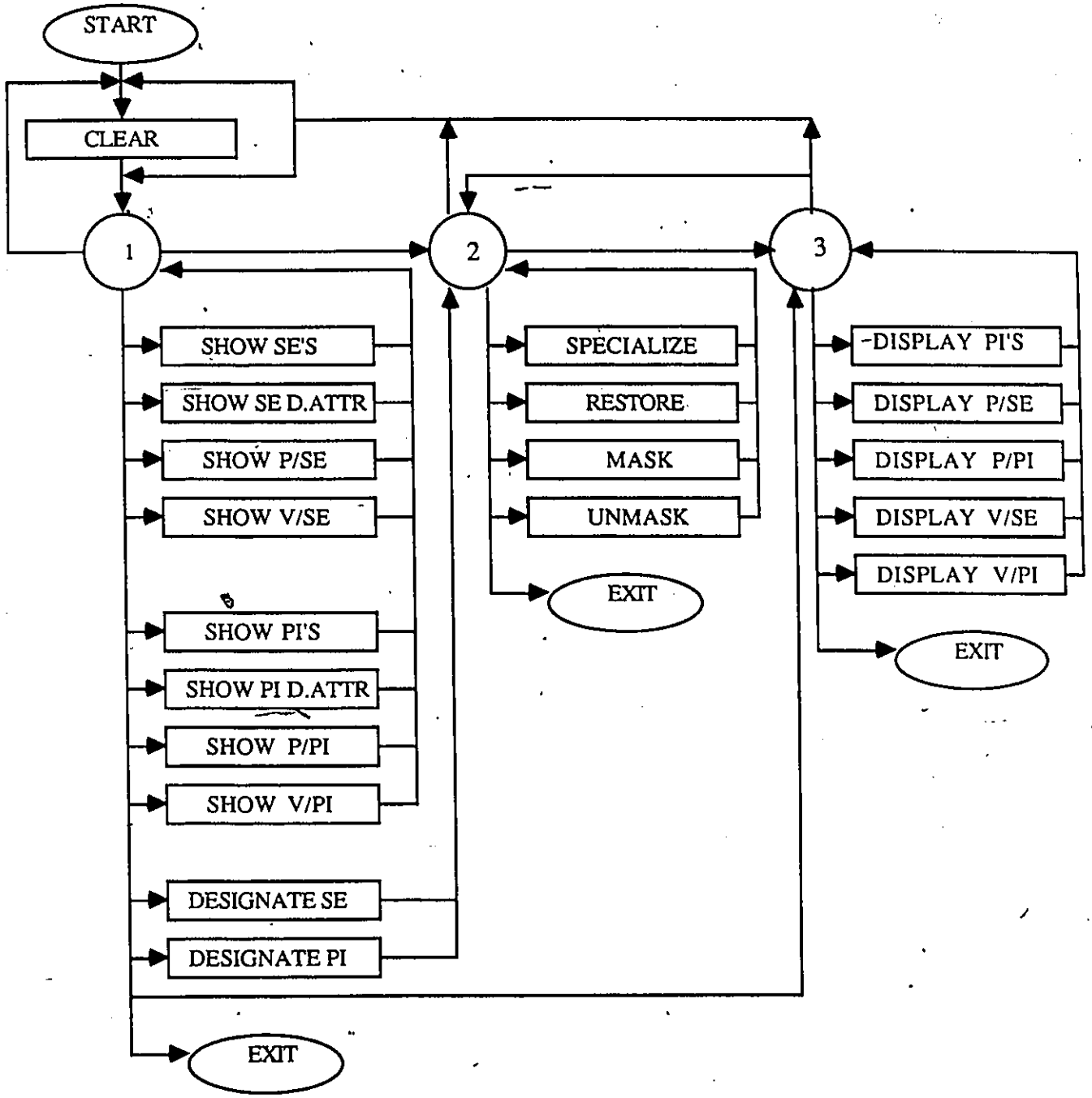


Figure 3.3

Structure of the Query Subsystem.

following sections provide some details about the activities related to these menus.

3.3.1.1 THE SHOW/DESIGNATE MENU

The SHOW/DESIGNATE menu provides the means for the user either to obtain information relating to the specifications of an SE or a PI, or to indicate, i.e. "designate", an SE and/or a PI which are of particular interest to him. The user has the following choices within the SHOW/DESIGNATE menu :

(a) list the names of all the SE's within the system;

(b) view the defining attributes of some specific SE;

(c) view the parameter specifications of some specific SE;

(d) view the variable specifications of some specific SE;

(e) list the names of all the PI's within the system;

(f) view the defining attributes of some specific PI;

(g) view the parameter specifications of some specific PI;

(h) view the variable specifications of some specific PI;

(i) designate a particular SE for further investigation;

(j) designate a particular PI for further investigation.

The SHOW/DESIGNATE menu (and in fact all three menus) provides a

display of the currently designated SE and/or PI. If no SE and/or PI has been designated, then the displays show "*NONE*". This, in particular, occurs when the query subsystem is first entered.

When the user designates a PI, an SE is also implicitly designated. This is because each PI in the system is bound to an "SE_master". If an SE has already been designated, then NEMS checks whether this SE is the SE_master of the PI. If the PI is not bound to this SE, then NEMS provides the user with the option of cancelling the designation of the PI, or of retaining the PI which then overwrites the currently designated SE with this PI's SE_master.

Similarly, if a PI has already been designated when the user designates an SE, NEMS checks whether this SE is the SE_master of the currently designated PI. If it is not the SE_master of the PI, then NEMS gives the user the option of cancelling the designation of the SE, or of retaining the SE. In this latter case, previously designated PI is "reset"; i.e. NEMS proceeds as though there was no designated PI.

When the user designates a PI and an SE, this means that the user is only interested in the experiments carried out with this specific SE and PI combination. Thereafter, NEMS only needs to manage the set of experiments carried out with this SE/PI combination. Similarly, when the

user designates an SE without designating a PI, the set of experiments that NEMS needs to manage is restricted to those carried out with this SE.

A convenient means used by NEMS to manage the experiments is to place those experiments that are of particular interest to the user in a set called the "active set". The active set, at any stage of the query activity, contains some subset of the tuples that are stored in the relation EXP_Lib. We refer to the total number of experiments, i.e. tuples, in this active set as NAS (Number in Active Set) and the value of NAS is displayed on each menu. When the user first enters the query subsystem, there is no designated SE or PI, and the NAS value that is displayed is the total number of experiments that are stored in the relation EXP_Lib.

After designating a specific SE, the active set is contracted to include only those experiments where this specific SE is used. Furthermore, if a PI is also designated, then the active set is further contracted to include only those experiments where the designated SE is used in conjunction with the designated PI.

Another means for contracting the active set is through "refinement". This refinement process is discussed in the following section.

Note that at any stage in the query process, the user can choose the CLEAR option to reset the designated SE and PI. When the designated SE

and PI are reset, the active set is automatically reset to include all experiments stored in EXP_Lib.

3.3.1.2 THE REFINE MENU

Once an SE or a PI has been designated, the user has the option of moving to the REFINE menu. The REFINE menu provides a means for restricting the set of experiments that the user regards as having current relevance. Two refinement categories are available, namely, specialization and masking.

The specialization action can be applied to both parameters and variables. The action involved here is that of specifying either a specific value or a range of values for any specific parameter or variable. The effect is to constrain the active set so that it references only those experiments where the specialized values occur.

When the user chooses the specialize option in the REFINE menu, another menu comes up. This secondary menu provides the user with the means to identify any set of parameters or variables within either a designated SE or a designated PI and then to assign to each, a specific value or range of values. After this specification, NEMS examines each of the experiments currently in the active set, and deletes those that do not

satisfy these prescribed specifications. In other words, the active set is contracted to contain references only to those experiments where the specialized values occur.

A second alternative in the REFINE menu is the mask option. Frequently, during the query activity, there is no interest in actually viewing the values of all the parameters or variables that are recorded for the experiments. Consequently, a means is provided to mask an arbitrary number of these parameters and variables so that during subsequent displays, their values are not presented. The user is thus able to focus his attention on those parameters and variables that are of special interest. Note that the masking operation does not affect the active set in any way.

The selection of the masking operation results in the display of a secondary menu. This menu provides the means to mask any specific parameters or variables in any designated SE or PI. NEMS then marks these identifiers as "masked" and any information related to them is not shown in any subsequent display.

Note that both the specialization and masking operations may be "undone". In the REFINE menu, the user is provided with options either to retract previously specified value/range specifications (the restore

option), or to unmask all the masked parameters or variables (the unmask option). When the user chooses the restore option, NEMS restores to the active set the references to those experiments that NEMS removed during the specialization process.

When the unmask option is chosen, an unmask operation menu comes up. The user is prompted for the type of identifiers that he wishes to unmask. The user can unmask identifiers by type only, i.e. all SE parameters, all SE variables, all PI parameters, or all PI variables. NEMS does not provide the facility to unmask individual identifiers.

3.3.1.3 THE DISPLAY MENU

The DISPLAY menu is accessible to the user after the designation of either an SE or a PI. Recall however that the designation of a PI implicitly designates an SE; namely, the SE_master of the PI. Hence, an SE will always be designated when in the DISPLAY menu. In the DISPLAY menu, the user can browse through the experimental results that are stored in the NEMS database. The data that can be referenced corresponds to the active set together with the influence of masking operations that may be in effect.

Within the DISPLAY menu, the user has the following options :

(a) list all the PI's that have been used with the currently designated SE;

(b) display the active values assigned to the unmasked SE parameters, for each of the experiments in the active set (these will correspond to the experiments carried out with the currently designated SE and the currently designated PI (if any), appropriately modified by the user's refinement specifications);

(c) display the active values assigned to the unmasked PI parameters, for each of the experiments in the active set;

(d) display the values acquired by the unmasked SE variables, for each of the experiments in the active set;

(e) display the values acquired by the unmasked PI variables, for each of the experiments in the active set.

Even with the restricting facility provided by the "refine" option, there may still be a large number of experiments referenced in the active set. Some additional flexibility in browsing through these is provided by allowing the user to skip over an arbitrary number of these experiments. After displaying each experiment, NEMS displays the total number of experiments which is yet to be displayed, and prompts the user for the number of experiments that he wishes to skip. Since the experiments are

displayed in a chronological order, this feature can, for example, allow the user to display the most recent experiment by simply skipping to the last experiment.

3.3.2 PROGRAM STRUCTURE FOR THE QUERY MODE

As indicated previously, there are three main menus provided in the query subsystem, namely, the SHOW/DESIGNATE menu, the REFINE menu and the DISPLAY menu. The menu flow of this subsystem is controlled by a routine called Q_Main which in turn, calls three different routines for this purpose. In addition, Q_Main can call a routine that resets the query subsystem. A representation of the program structure of the query subsystem is given in Figure 3.4, and a description of the programs is given below :

(a) Q_Main : controls the menu flow of the query subsystem. It calls the routines listed in (b), (c), (d) and (e).

(b) Q_Clear : resets the query subsystem.

(c) Q_Menu1 : controls the SHOW/DESIGNATE menu. Q_Menu1 calls the routine Q_SD which performs the show/designate operations. Q_SD in turn calls two routines Q_SE and Q_PI. Q_SD calls Q_SE if the user wishes either to view the specifications relating to an SE, or to designate an SE.

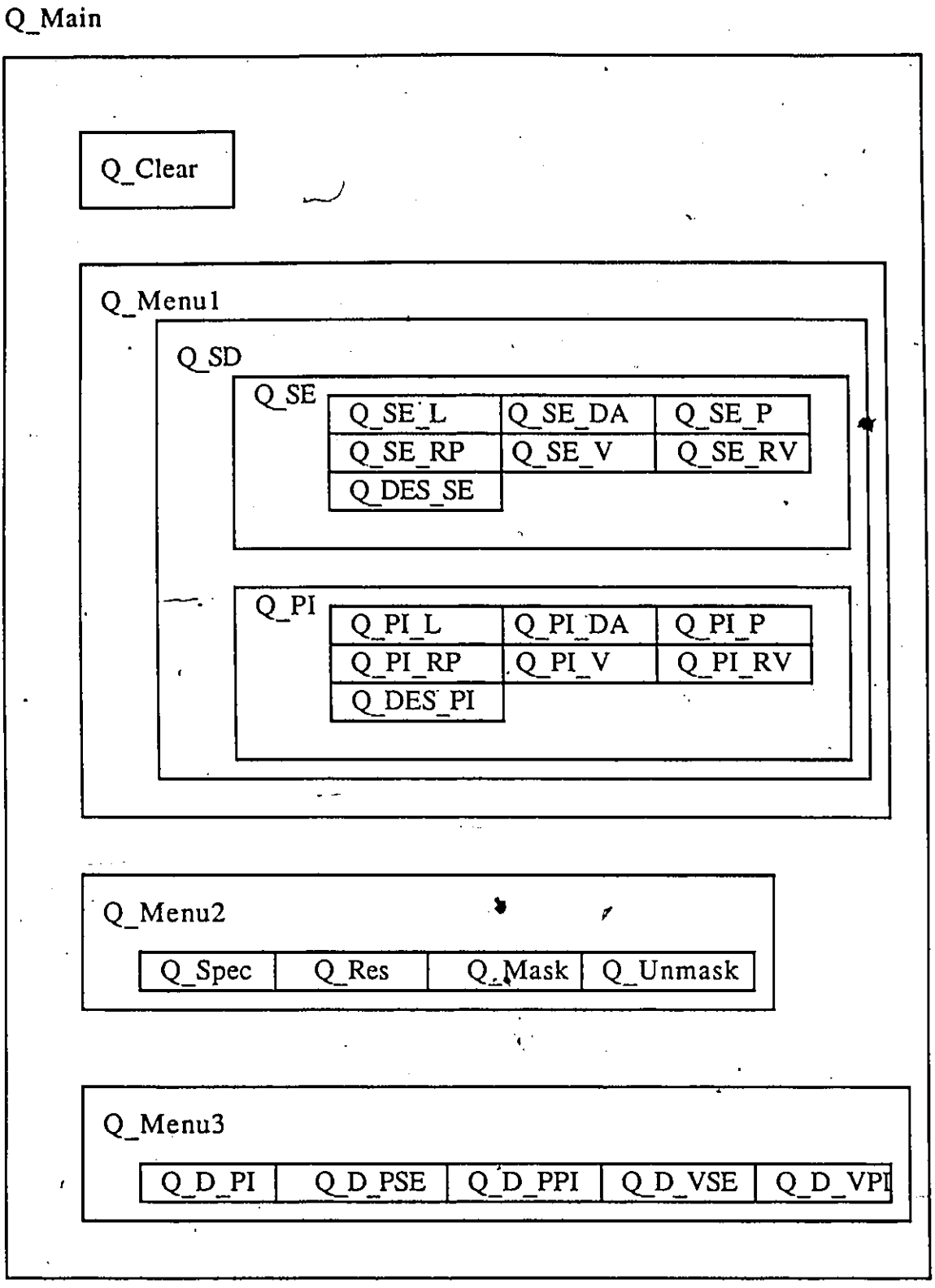


Figure 3.4

— Program Structure for the Query Mode.

Q_SE calls the routines Q_Des_SE to perform the designation, Q_SE_L to list all the SE's in NEMS, Q_SE_DA to display the defining attributes of the SE, Q_SE_P to display SE parameter specifications, Q_SE_RP to display SE reflected parameter specifications, Q_SE_V to display SE variable specifications, and Q_SE_RV to display SE reflected variable specifications. Similarly, Q_SD calls Q_PI if the user wishes to view the specifications relating to a PI, or designate a PI. Q_PI calls the routines Q_Des_PI to perform the designation, Q_PI_L to list all the PI's in NEMS, Q_PI_DA to display the defining attributes of the PI, Q_PI_P to display PI parameter specifications, Q_PI_RP to display reflected parameter specifications, Q_PI_V to display PI variable specifications, and Q_PI_RV to display reflected variable specifications.

(d) Q_Menu2 : controls the REFINE menu. Q_Menu2 calls the routines Q_Spec, Q_Res, Q_Mask, Q_Unmask to perform the various refinement operations; namely, specialization, restoring from specialization, masking parameters and variables, and unmasking parameters and variables.

(e) Q_Menu3 : controls the DISPLAY menu (recall that during the DISPLAY activity, the parameter and variable values which are displayed relate only to those experiments that are in the current active set). Q_Menu3 calls the routines Q_D_PI to list all the PI's that are bound to

the currently designated SE, Q_D_PSE to display the active parameter values of the currently designated SE for each experiment in the active set, Q_D_PPI to display the active parameter values of the currently designated PI for all the experiments in the active set, Q_D_VSE to display the variable values of the currently designated SE for each experiment in the active set, and Q_D_VPI to display the variable values of the currently designated PI for each experiment in the active set.

3.4 THE DELETE MODE

As outlined in Section 2.3.4, the user can delete an SE or a PI from the NEMS database in the delete mode. When an SE or a PI is deleted, all information relating to it is also deleted. In the case of the deletion of an SE, this in particular, implies that each PI having the SE as its master will also be deleted. A representation of the delete mode is given Figure 2.4 and some implementational aspects are presented in the following sections.

3.4.1 DELETION OF AN SE

Suppose the user wishes to delete an SE whose name is SE_NAME and suppose furthermore that the NEMS internal identifier for SE_NAME is SE<nn>. The deletion of SE_NAME involves the following steps :

(a) Delete from the relation SE_Lib that tuple whose SE_CODE attribute value is SE<nn>.

(b) Delete from the relation EXP_Lib each tuple whose SE_CODE attribute value is SE<nn>.

(c) Delete the relations PSE<nn>, RPSE<nn>, VSE<nn>, RVSE<nn>, ESE<nn>, SE<nn>I<ii>, SE<nn>P<ii>, SE<nn>V<ii> and SE<nn>C<ii>, as appropriate.

(d) For each PI having SE_NAME as its master,

(i) delete from the relation PI_Lib the tuple whose PI_CODE attribute value is PI<nn>, where PI<nn> is the internal NEMS identifier for the PI.

(ii) delete the relations PPI<nn>, VPI<nn>, EPI<nn>, PI<nn>I<ii>, PI<nn>P<ii>, PI<nn>V<ii> and PI<nn>C<ii> as appropriate.

3.4.2 DELETION OF A PI

Suppose the user wishes to delete a PI whose name is PI_NAME, and suppose furthermore that the NEMS internal identifier for PI_NAME is PI<nn> and that the NEMS internal identifier for the SE_master for PI_NAME is SE<nn>. The deletion of PI_NAME involves the following steps :

(a) Delete from the relation PI_Lib the tuple whose PI_CODE attribute value is PI<nn>.

(b) Delete the relations PPI<nn>, VPI<nn>, EPI<nn>, PInn>I<ii>, PI<nn>P<ii>,

PI<nn>V<ii> and PI<nn>C<ii>, as appropriate.

(c) Delete each tuple from the relation EXP_Lib which has a PI_CODE attribute value of PI<nn>.

(d) Corresponding to each tuple deleted in (c), delete from each of the relations ESE<nn>, SE<nn>P<ii>, SE<nn>V<ii> and SE<nn>C<ii> the tuple whose SE_SEQ_NUM attribute value matches that of the tuple deleted in (c).

3.4.3 PROGRAM STRUCTURE FOR THE DELETE MODE

Figure 3.5 gives a representation of the structure of the program module that relates to delete mode. The various programs that comprise this module are outlined below :

(a) Del : carries out the deletion of an SE or a PI. It calls the routines which are listed in (b) and (c).

(b) Del_SE : carries out the deletion of an SE whose NEMS internal identifier is SE<nn>. Del_SE in turn, calls the routines Del_SLR which carries out three tasks; namely, (i) deletes from the relation SE_Lib the tuple which stores the defining attributes of SE<nn>, (ii) deletes from the relation EXP_Lib all the experiments which used SE<nn>, and (iii) deletes the relations PSE<nn>, RPSE<nn>, VSE<nn>, RVSE<nn>, ESE<nn>, SE<nn>I<ii>, SE<nn>P<ii>, SE<nn>V<ii> and SE<nn>C<ii> as appropriate. Del_SE also calls the

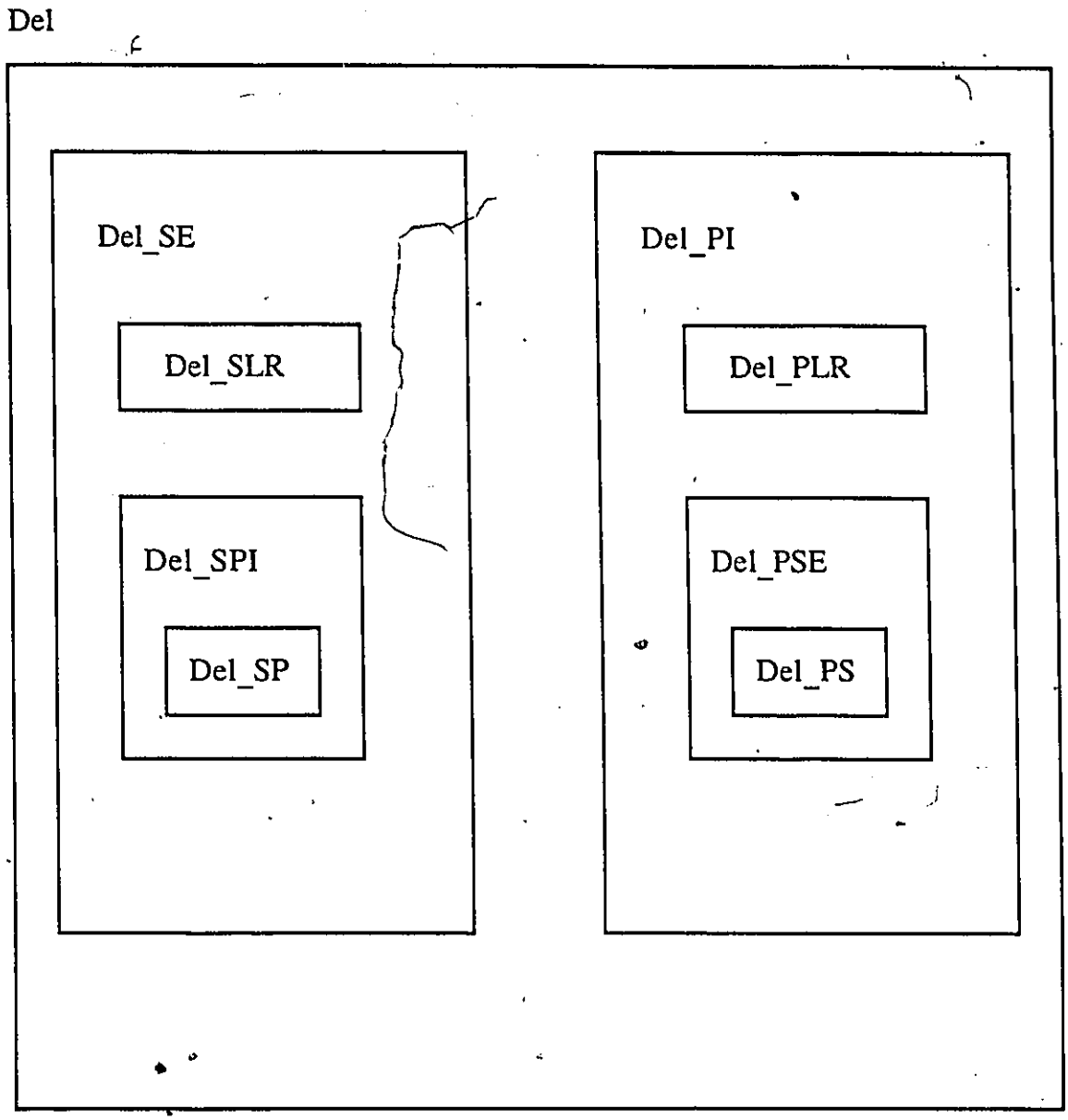


Figure 3.5

Program Structure for the Delete Mode.

routine Del_SPI, which in turn calls the routine Del_SP which performs the deletion of each PI having SE<nn> as its master. This involves the deletion of the PI from the relation PI_Lib and the deletion of all relations associated with the PI; i.e. PPI<nn>, VPI<nn>, EPI<nn>, PI<nn>I<ii>, deletion of each PI having SE<nn> as its master. This involves the deletion of the PI from the relation PI_Lib and the deletion of all relations PI<nn>P<ii>, PI<nn>V<ii> and PI<nn>C<ii> as appropriate.

(c) Del_PI : carries out the deletion of a PI whose NEMS internal identifier is PI<nn> and whose SE_master is SE<nn>. Del_PI in turn, calls the routines Del_PLR which deletes from the relation PI_Lib, the tuple which stores the defining attributes of PI<nn>, and also deletes the relations PPI<nn>, VPI<nn>, EPI<nn>, PI<nn>I<ii>, PI<nn>P<ii>, PI<nn>V<ii> and PI<nn>C<ii> as appropriate. Del_PI also calls the routine Del_PSE which deletes from the relation EXP_Lib, all experiments which used PI<nn>. In addition, Del_PSE also calls the routine Del_PS which deletes from the relations ESE<nn>, SE<nn>P<ii>, SE<nn>V<ii> and SE<nn>C<ii> the tuples that resulted from experiments which used PI<nn>.

CHAPTER 4

AN EXAMPLE

4.1 INTRODUCTION

In this chapter we illustrate the use and the capabilities of NEMS by considering a representative solution engine and a related problem instance. Following a brief description of the specific solution engine being considered, the procedure for incorporating this SE within NEMS is outlined, and the steps involved in utilizing this SE are summarized. Using some representative data, the NEMS query mode is also illustrated.

4.2 A REPRESENTATIVE SOLUTION ENGINE

As the focal point for demonstrating NEMS, we have chosen the DIFPAK package [5]. DIFPAK is an integrated collection of FORTRAN subprograms which has been designed to generate the numerical solution of a system of first order ordinary differential equations; i.e. a system of equations of the form

$$\dot{x}(t) = f(t, x(t)) \qquad (*)$$

where x (the state vector) is a vector of length n. In the basic mode, it

provides an easy-to-use and effective means for obtaining the numerical solution of (*) over a prescribed interval, from a specified initial condition, x(t0); i.e. the classical initial value problem. However, a highly modular approach has been used in developing the package in order to make it rapidly adaptable to a variety of different problem environments. Included here are two-point boundary value problems and optimization problems in which optimal values are desired for parameters that are imbedded within the function f on the right-hand side of (*).

The modularity provided in DIFPAK has also been formulated from the point of view of facilitating its use in investigating novel schemes for handling "difficult" problems; e.g. stiff equations and systems with severe discontinuities. Strategies involving the use of variable order integration methods can also be easily accommodated. The package can, therefore, serve as a research tool and has, in fact, been used as a key module in the studies described in [6,7,8,9].

Twelve numerical integration methods are currently provided in DIFPAK and these include members of both the Runge-Kutta family and the predictor-corrector family. These methods are all fundamentally of

the variable step-size type, but each may, if desired, be invoked in a fixed step-size mode. The specification of the particular method to be used in any particular application, is controlled by the DIFPAK parameter called, METHOD. DIFPAK consists of approximately 2000 lines of FORTRAN code and is supported at the University of Ottawa by a VM/CMS EXEC procedure which facilitates its use on the University's AMDAHL mainframe. All arithmetic operations are carried out in the double precision mode.

The principal output produced by DIFPAK is the solution to (*) represented as a sequence of "points" along the t-axis; i.e., along the axis of the problem independent variable. Each such "point" is, in fact, a collection of (n+1) data values represented by the value of t and the values of n state variables, x(i), i = 1,2,.....,n. This principal DIFPAK output therefore corresponds to our notion of a cluster-list.

DIFPAK contains a relatively large number of parameters and an attempt to accommodate all of them in the present context would obscure the intended illustration of NEMS. A slightly specialized version of DIFPAK has, therefore, been formulated for which many of the DIFPAK parameters are hidden from the user and assigned default values. The

parameters that have been retained are as follows :

(a) METHOD: As indicated previously, this scalar parameter serves to specify the desired solution method. An integer coding scheme is used; i.e. METHOD is an integer parameter.

(b) VERIFY : DIFPAK is frequently used to test the effectiveness of a newly proposed solution method. In such circumstances it is useful to obtain a comparison between the solution generated by the candidate method and a "true" solution. This true solution may be either from an explicit analytical formula or from a highly accurate numerical procedure. The VERIFY parameter serves as a toggle switch to turn this comparison feature "on" (VERIFY = 1) or "off" (VERIFY = 0). VERIFY is an integer parameter.

(c) ABSERR : The step-size adjustment procedure in DIFPAK seeks to maintain an estimated local truncation error that is no worse than a user provided specification. ABSERR is one of two parameters which control this latter specification. ABSERR has a double precision value.

(d) RELERR : This parameter is the second one that is involved in the

user specification indicated in (c) above. RELERR has a double precision value.

(e) HSTRT : The integration step-size procedure requires an initial value for the step-size which is subsequently adjusted as necessary. The HSTRT parameter provides this value in a double precision mode.

A "problem" from the point of view of DIFPAK, is a specific set of first order ordinary differential equations of the form shown in (*). Such a problem has a set of intrinsic attributes which must be known to DIFPAK. Since every "DIFPAK-directed" problem must have these attributes; they are regarded as reflected parameters of DIFPAK itself. These attributes (i.e., reflected parameters) are :

(a) N : The number of first order differential equations to be solved (an integer value).

(b) FINTIM : The right-hand end of the solution interval over which the problem is to be solved (a double precision value).

(c) XZ : The vector of initial conditions associated with the problem; i.e., the value of x(t0) (each component if a double precision value).

As noted previously, the fundamental part of the output generated by a DIFPAK execution is a sequence of solution values (a cluster-list). We note, however, that this output properly belongs not to DIFPAK itself, but to the specific problem that is being solved. In other words the cluster-list that is the generated solution to the problem is a reflected variable of DIFPAK. In the sequel we assign the name TRAJ to this particular clist variable.

Two non-reflected DIFPAK variables do, however, exist. These are :

(a) NRHS : This provides a count of the number of evaluations of the derivative function, f, (the right-hand-side of (*)) required over the solution interval (a scalar integer value).

(b) AVG : This provides the average integration step-size over the solution interval (a scalar double precision value).

As a representative problem instance for DIFPAK we consider the following set of first-order ordering differential equations :

$$\dot{x}1(t) = - x2(t) - x1(t) * x3(t) / R(t)$$

$$\dot{x}2(t) = x1(t) - x2(t) * x3(t) / R(t)$$

$$\dot{x}3(t) = x1(t) / R(t)$$

where $R(t) = \sqrt{x1(t)**2 + x2(t)**2}$

The formal realization of any problem instance for DIFPAK takes the form of a set of FORTRAN subprograms. This set always contains at least one member; namely, the subroutine subprogram called RHS which provides the specifications for the differential equations that are to be studied. The specific RHS subroutine for the problem under consideration is shown below :

```
      SUBROUTINE RHS(T , X , F)
      IMPLICIT  REAL*8  (A-H, O-Z)
      REAL*8  X(1) , F(1)
      R = DSQRT(X(1)**2 + X(2)**2)
      F(1) = -X(2) - X(1) * X(3) / R
      F(2) = X(1) - X(2) * X(3) / R
      F(3) = X(1) / R
      RETURN
      END
```

For "simple" problems, such as the one being considered, the RHS subprogram is the only one that is required for the realization of the problem instance.

The parameters of any problem instance for DIFPAK will necessarily include the three reflected parameters defined for DIFPAK : namely, N, FINTIM and XZ. The value of N in the present case is three and it is clearly not meaningful to assign any other value. The default value chosen for FINTIM is 3.0 and the default initial condition vector is taken to be (3.0 , 0 , 0). The present problem instance has no other parameters apart from these reflected parameters.

The only variable associated with this problem instance is the DIFPAK reflected variable called TRAJ which is a cluster-list (every DIFPAK related problem instance inherits this particular variable). In the present case, the size of this cluster-list is four (i.e. the value of t, the problem independent variable and the corresponding values of the three state variables).

4.3 INITIATING NEMS

The system configuration upon which the use of NEMS is based, includes the following :

(a) a PC running MS/DOS, with 512K RAM and a hard disk;

(b) a directory partition containing dBASE III, TURBO PASCAL, a file

transfer program and the NEMS module itself;

(c) a mainframe computer system which is accessible from the communication software noted in (b) and which maintains appropriate program files for the solution engines and problem instances that are to be utilized under NEMS;

(d) a named "EXEC file" associated with each solution engine that is resident on the mainframe whose purpose is to appropriately activate the program code of the solution engine and its associated problem instances.

The NEMS module exists as a set of dBASE III procedures and relations. In order to initiate a NEMS session, the user first activates dBASE III and then enters the command

"DO NEMS"

This results in the display of the main menu shown in Figure 4.1. The user has now entered NEMS, and can choose either to activate any of the four NEMS modes; namely, the create mode, the execute mode, the query mode or the delete mode, or to exit NEMS.

4.4 ILLUSTRATION OF THE CREATE MODE

We assume in our example that the user wishes to create the

solution engine "DIFPAK", hence the option "CREATE SE/PI" is selected from the main menu (Figure 4.1). This results in the display of another menu (Figure 4.2) from which the user chooses the "CREATE SE" option. The system then undertakes to obtain from the user the defining attributes of the solution engine. This is achieved by appropriately "filling in the blanks" on a subsequent screen as shown in Figure 4.3.

As the next step, the system prompts the user for the specifications for each of the SE parameters. An initial screen for this purpose is shown in Figure 4.4. Following the selection of "1" for CHOICE, mode information is requested in the format shown in Figure 4.5. Then, because the parameter type is scalar numeric, the system requests upper bound, lower bound and default value information in the format shown in Figure 4.6. These values may be left unassigned, but a blank for a default value is subsequently interpreted as a "0" for a numeric type. This process is repeated for each of the SE parameters. In the case of DIFPAK, the parameters VERIFY, ABSERR, RELERR and HSTRT would be declared, in addition to METHOD.

Following completion of the parameter specification for each of the parameters, the system prompts the user for the specifications for each of

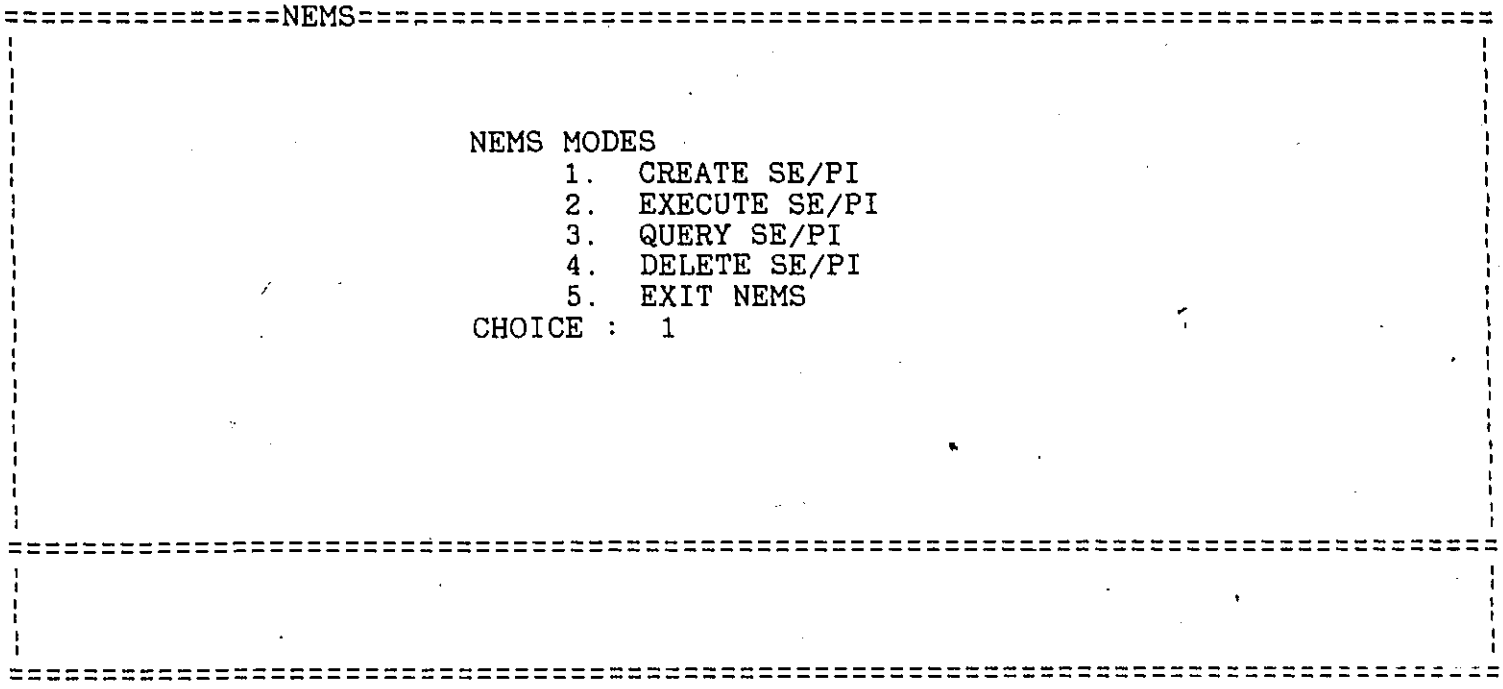


Figure 4.1

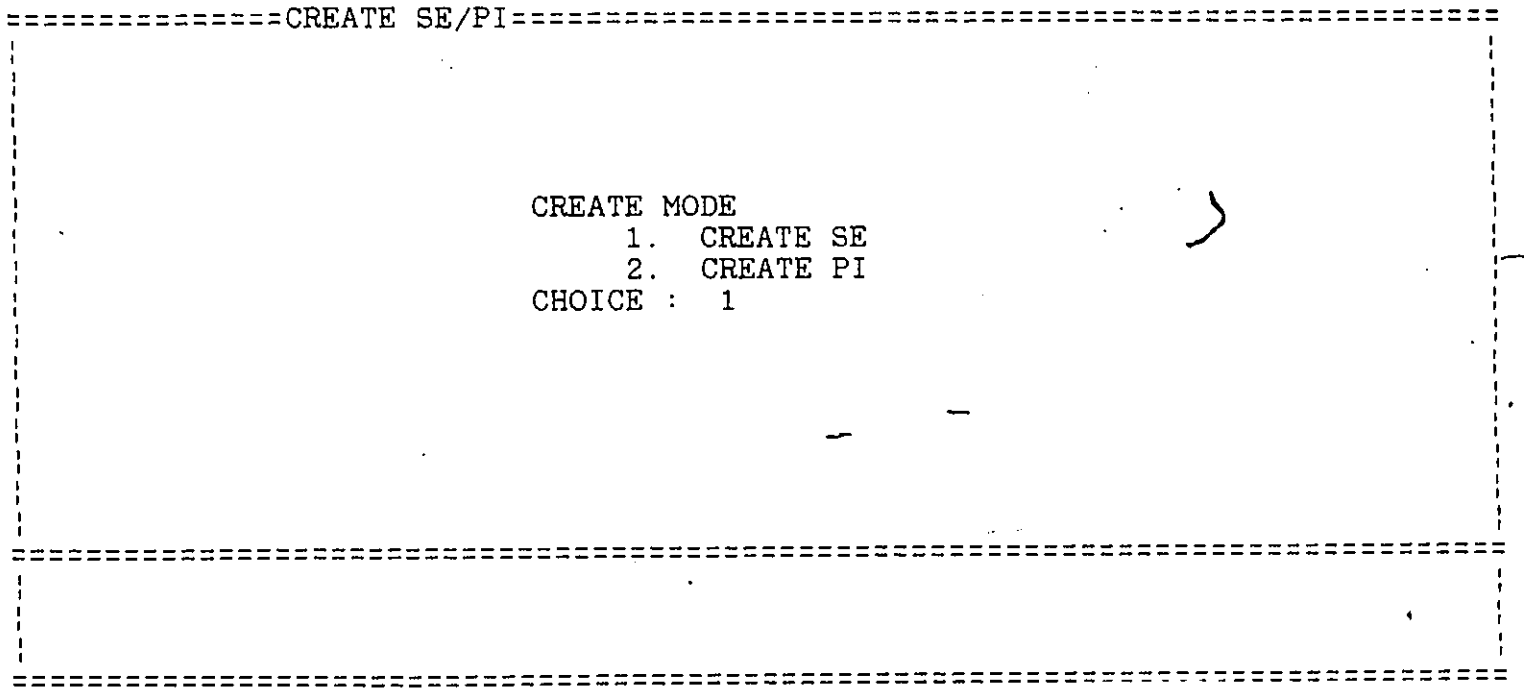


Figure 4.2

```
==============CREATE SE=========================================================
|                                                                              |
|                                                                              |
|                NAME OF SOLUTION ENGINE (SE) :  DIFPAK                        |
|                NUMBER OF PARAMETERS :   5                                    |
|                NUMBER OF REFLECTED PARAMETERS :   3                          |
|                NUMBER OF VARIABLES :   2                                     |
|                NUMBER OF REFLECTED VARIABLES :   1                           |
|                NAME OF EXEC FILE WHICH EXECUTES THE SE :  DIFNEMS             |
|                PRECISION (SINGLE/DOUBLE) :  D                                |
|                OWNER :  MARIA                                                |
|                                                                              |
|                                                                              |
==============================================================================|
|                                                                              |
|                        ENTER VALUES AND PRESS RETURN...                      |
|                                                                              |
==============================================================================|
```

Figure 4.3

```
==============CREATE SE=========================================================
| SOLUTION ENGINE : DIFPAK                                                     |
|                                                                              |
|        PARAMETER 1 :                                                         |
|        ------------                                                          |
|                  NAME :  METHOD                                              |
|                  TYPE :                                                      |
|                     1. SCALAR NUMERIC                                        |
|                     2. VECTOR NUMERIC                                        |
|                     3. CHARACTER STRING                                      |
|                  CHOICE :  1                                                 |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
==============================================================================|
|                                                                              |
|                        ENTER VALUES AND PRESS RETURN...                      |
|                                                                              |
==============================================================================|
```

Figure 4.4

```
===============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

         PARAMETER 1 :
         ------------
                    NAME :  METHOD
                    TYPE :
                       1. SCALAR NUMERIC
                       2. VECTOR NUMERIC
                       3. CHARACTER STRING
                    CHOICE :  1


                    ENTER MODE OF SCALAR NUMERIC :
                          1.  INTEGER
                          2.  REAL
                    CHOICE :  1
=====================================================================

                         ENTER A NUMBER AND PRESS RETURN...

=====================================================================
```

Figure 4.5

```
===============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

         PARAMETER 1 :
         ------------
                    NAME :  METHOD
                    TYPE :
                       1. SCALAR NUMERIC
                       2. VECTOR NUMERIC
                       3. CHARACTER STRING
                    CHOICE :  1

              LOWER              UPPER              DEFAULT
              BOUND              BOUND              VALUE
              -----              -----              -------
              0                  100                9

=====================================================================


=====================================================================
```

Figure 4.6

the reflected parameters. The user enters "N" as the name of the first parameter and specifies its type as "SCALAR NUMERIC" (Figure 4.7). The mode of N is then specified as "INTEGER" (Figure 4.8) and this completes the specification of the reflected parameter N.

The specifications for the reflected parameter FINTIM are entered in the same way as those of N, except that the mode of FINTIM is specified as "REAL".

The third reflected parameter is the real vector XZ. The user enters XZ as the name of the reflected parameter, chooses the type "VECTOR NUMERIC" (Figure 4.9), and the mode "REAL" (Figure 4.10). Because of the vector type of XZ, the user is prompted for the maximum length of XZ, and the value "10" is entered (Figure 4.11). The system then undertakes the determination of the active length for XZ. This begins with the query shown in Figure 4.12. Since the active length of XZ is not fixed, "N" (NO) is entered. Nor is the active length to be explicitly assigned by the user at execution time, hence "N" is entered as the response to the next query as shown in Figure 4.13. This implies that the active length of XZ is via another parameter or variable. Therefore, the user is prompted for the name of the source identifier for the active length. Since the active

```
===============CREATE SE=========================================
  SOLUTION ENGINE : DIFPAK

         REFLECTED PARAMETER 1 :
         ----------------------
                   NAME :  N
                   TYPE :
                      1. SCALAR NUMERIC
                      2. VECTOR NUMERIC
                      3. CHARACTER STRING
                   CHOICE :  1

=================================================================

                        ENTER VALUES AND PRESS RETURN...

=================================================================
```

Figure 4.7

```
===============CREATE SE=========================================
  SOLUTION ENGINE : DIFPAK

         REFLECTED PARAMETER 1 :
         ----------------------
                   NAME :  N
                   TYPE :
                      1. SCALAR NUMERIC
                      2. VECTOR NUMERIC
                      3. CHARACTER STRING
                   CHOICE :  1

                   ENTER MODE OF SCALAR NUMERIC :
                         1.  INTEGER
                         2.  REAL
                   CHOICE :  1
=================================================================

                       ENTER A NUMBER AND PRESS RETURN...

=================================================================
```

Figure 4.8

```
==============CREATE SE=========================================================
  SOLUTION ENGINE : DIFPAK

         REFLECTED PARAMETER 3 :
         ----------------------
                   NAME :  XZ
                   TYPE :
                      1. SCALAR NUMERIC
                      2. VECTOR NUMERIC
                      3. CHARACTER STRING
                   CHOICE :  2

================================================================================

                          ENTER VALUES AND PRESS RETURN...

================================================================================
```

Figure 4.9

```
==============CREATE SE=========================================================
  SOLUTION ENGINE : DIFPAK

         REFLECTED PARAMETER 3 :
         ----------------------
                   NAME :  XZ
                   TYPE :
                      1. SCALAR NUMERIC
                      2. VECTOR NUMERIC
                      3. CHARACTER STRING
                   CHOICE :  2


                   ENTER MODE OF VECTOR NUMERIC :
                         1. INTEGER
                         2. REAL
                   CHOICE :  2
================================================================================

                        ENTER A NUMBER AND PRESS RETURN...

================================================================================
```

Figure 4.10

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED PARAMETER 3 :
        -----------------------
                  NAME :  XZ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CHARACTER STRING
                  CHOICE :  2


                  ENTER THE MAXIMUM LENGTH OF VECTOR :  10
```

Figure 4.11

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED PARAMETER 3 :
        -----------------------
                  NAME :  XZ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CHARACTER STRING
                  CHOICE :  2

            IS THE ACTIVE LENGTH OF XZ FIXED? (Y/N)  N
```

Figure 4.12

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED PARAMETER 3 :
        -----------------------
                  NAME :  XZ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CHARACTER STRING
                  CHOICE :  2


        IS ACTIVE LENGTH OF XZ USER ASSIGNABLE? (Y/N)  N
```

Figure 4.13

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED PARAMETER 3 :
        -----------------------
                  NAME :  XZ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CHARACTER STRING
                  CHOICE :  2


        ENTER NAME OF SOURCE IDENTIFIER :  N
```

Figure 4.14

length of XZ is via the reflected parameter N, the user enters the name "N" (Figure 4.14). This ends the specifications for the reflected parameter XZ.

After the specifications for the SE reflected parameters, the user is prompted for the specifications for the SE variables. The user enters "NRHS" as the name of the first such variable and chooses "SCALAR NUMERIC" as its type (Figure 4.15) and "INTEGER" as its mode (Figure 4.16). The specifications for the other variable, AVG, are entered in the same way; however, its mode is chosen as "REAL".

After the specifications of variables, the user is prompted for the specifications for the reflected variables. There is only one such variable in this example and it is a cluster list. The user enters "TRAJ" as the name of this reflected variable, chooses "CLUSTER LIST" as its type (Figure 4.17), and "REAL" as its mode (Figure 4.18). Now the user is prompted for the specifications for the active size of TRAJ. Since the size of TRAJ is fixed, the answer "N" is entered in response to the query of Figure 4.19. The user is then asked whether the size of TRAJ is user assignable, and the response "Y" (YES) is entered (Figure 4.20). This ends the specifications for the reflected variable TRAJ.

```
==============CREATE SE=========================================================
  SOLUTION ENGINE : DIFPAK

          VARIABLE 1 :
          -----------
                     NAME :  NRHS
                     TYPE :
                        1. SCALAR NUMERIC
                        2. VECTOR NUMERIC
                        3. CLUSTER LIST
                        4. CHARACTER STRING
                     CHOICE :  1
================================================================================

                          ENTER VALUES AND PRESS RETURN...

================================================================================
```

Figure 4.15

```
==============CREATE SE=========================================================
  SOLUTION ENGINE : DIFPAK

          VARIABLE 1 :
          -----------
                     NAME :  NRHS
                     TYPE :
                        1. SCALAR NUMERIC
                        2. VECTOR NUMERIC
                        3. CLUSTER LIST
                        4. CHARACTER STRING
                     CHOICE :  1

                     ENTER MODE OF SCALAR NUMERIC :
                           1.  INTEGER
                           2.  REAL
                     CHOICE :  1
================================================================================

                          ENTER A NUMBER AND PRESS RETURN...

================================================================================
```

Figure 4.16

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED VARIABLE 1 :
        ----------------------
                  NAME :  TRAJ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CLUSTER LIST
                     4. CHARACTER STRING
                  CHOICE :  3

=====================================================================

                        ENTER VALUES AND PRESS RETURN...

=====================================================================
```

Figure 4.17

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED VARIABLE 1 :
        ----------------------
                  NAME :  TRAJ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CLUSTER LIST
                     4. CHARACTER STRING
                  CHOICE :  3

                  ENTER NUMERIC MODE OF CLUSTER LIST :
                        1.  INTEGER
                        2.  REAL
                  CHOICE :  2
=====================================================================

                       ENTER A NUMBER AND PRESS RETURN...

=====================================================================
```

Figure 4.18

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED VARIABLE 1 :
        ----------------------
                  NAME :  TRAJ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CLUSTER LIST
                     4. CHARACTER STRING
                  CHOICE :  3

                  IS SIZE OF TRAJ FIXED? (Y/N)  N
```

Figure 4.19

```
==============CREATE SE=============================================
  SOLUTION ENGINE : DIFPAK

        REFLECTED VARIABLE 1 :
        ----------------------
                  NAME :  TRAJ
                  TYPE :
                     1. SCALAR NUMERIC
                     2. VECTOR NUMERIC
                     3. CLUSTER LIST
                     4. CHARACTER STRING
                  CHOICE :  3

             IS SIZE OF TRAJ USER ASSIGNABLE? (Y/N)  Y
```

Figure 4.20

Based on the defining attribute values for DIFPAK as entered in Figure 4.3, the "definition" of this particular solution engine is complete at this point. NEMS therefore re-displays the main menu of Figure 4.1.

Suppose the user now wishes to create a problem instance called "EX1". He chooses the "CREATE SE/PI" option from the main menu, and on the subsequent menu (Figure 4.2), the user chooses the option "CREATE PI".

The user is then prompted for the defining attributes of the PI (Figure 4.21). Since there are no parameters or variables other than the reflected parameters and variables, the user enters "0" for both the number of parameters and the number of variables.

NEMS now proceeds to obtain lower bound, upper bound and default values for each problem instance parameter. In the case of EX1, the only parameters are reflected parameters. The user provides this information on the following screen (Figure 4.22) for each of the scalar reflected parameters. (Note that since there is no upper bound for the right-hand end of the solution interval, the upper bound for FINTIM is left blank.)

Each vector parameter is treated as a separate unit since lower bound, upper bound and default values must be obtained for each

```
==============CREATE PI=========================================================

                NAME OF PROBLEM INSTANCE (PI) :  EX1
                NAME OF SE MASTER :  DIFPAK
                NUMBER OF PARAMETERS :   0
                (in addition to refl. par)
                NUMBER OF VARIABLES :   0
                (in addition to refl. var)
                PRECISION (SINGLE/DOUBLE) :  D
                OWNER :  MARIA

================================================================================

                        ENTER VALUES AND PRESS RETURN...

================================================================================
```

Figure 4.21

```
==============CREATE PI=========================================================
   PROBLEM INSTANCE : EX1

   REFLECTED                    LOWER          UPPER          DEFAULT
   PARAMETER        TYPE        BOUND          BOUND          VALUE
   ---------        ----        -----          -----          -------

   N                S/INT  3                   3              3
   FINTIM           S/REAL 0                                  2.0

================================================================================

================================================================================
```

Figure 4.22

component. This information for the reflected vector parameter XZ is entered via the screen format shown in Figures 4.23 and 4.24. Because the maximum length of XZ was declared to be ten in the definition of DIFPAK, ten component numbers are displayed. However because the problem instance EX1 has only three state variables, only the first three components of XZ are meaningful.

Since EX1 has no non-reflected variables, the definition of the problem instance called EX1 terminates with the completion of the specifications for its parameters.

4.5 ILLUSTRATION OF THE EXECUTE MODE

When the user chooses the "EXECUTE SE/PI" option from the NEMS main menu, the secondary menu shown in Figure 4.25 is displayed, and the user is prompted for the names of the SE and PI to be used in the experiment. After these are identified, the system prompts the user for the active values of the scalar parameters of the solution engine, i.e. DIFPAK. (Figure 4.26). Note that any parameter that is not explicitly assigned a value by the user, is given its default value by the system.

After assignment of an active value to each DIFPAK parameter, the system prompts the user for the active values of scalar parameters of the

```
==============CREATE PI=============================================
  PARAMETER (REFLECTED)
      NUMBER : 3
      NAME   : XZ                    TYPE  : V/REAL

       COMPONENT    LOWER          UPPER          DEFAULT
       NUMBER       BOUND          BOUND          VALUE
       ---------    -----          -----          -------
           1                                       3
           2                                       0
           3                                       0
           4
           5
           6
           7
           8
====================================================================

                    ENTER VALUES AND PRESS RETURN.
                    MORE ON NEXT SCREEN...
====================================================================
```

Figure 4.23

```
==============CREATE PI=============================================
  PARAMETER (REFLECTED)
      NUMBER : 3
      NAME   : XZ                    TYPE  : V/REAL

       COMPONENT    LOWER          UPPER          DEFAULT
       NUMBER       BOUND          BOUND          VALUE
       ---------    -----          -----          -------
           9
          10
====================================================================

====================================================================
```

Figure 4.24

```
===============EXECUTE=====================================================
|                                                                         |
|                                                                         |
|                 NAME OF SOLUTION ENGINE :  DIFPAK                        |
|                                                                         |
|                 NAME OF PROBLEM INSTANCE:  EX1                           |
|                                                                         |
|                                                                         |
===========================================================================
|                                                                         |
|                                                                         |
===========================================================================
```

Figure 4.25

```
===============EXECUTE=====================================================
|   SOLUTION ENGINE : DIFPAK                                              |
|                                                                         |
|                                          DEFAULT          DESIRED        |
|   PARAMETER         TYPE                 VALUES           VALUES         |
|   ---------         ----                 -------          -------        |
|                                                                         |
|   METHOD            INTEGER NUMERIC      9                7              |
|   VERIFY            INTEGER NUMERIC      0                0              |
|   ABSERR            REAL NUMERIC         0.00001D0        0.00001D0      |
|   RELERR            REAL NUMERIC         0.00001D0        0.00001DQ      |
|   HSTRT             REAL NUMERIC         0.10000D0        0.10000D0      |
|                                                                         |
|                                                                         |
===========================================================================
|                                                                         |
|                                                                         |
===========================================================================
```

Figure 4.26

PI; namely, EX1 (Figure 4.27). The vector parameter, XZ, is handled on a component by component basis on a separate screen (Figure 4.28). Recall that during the creation process, the active length of XZ was specified as "via the active value of the scalar parameter N". Since the active value of N is three, the active length of XZ for this experiment becomes three. Hence, the system prompts the user for active values for only the first three components of XZ.

In general, after the assignment of the active values for the parameters of the PI, the user may be prompted for information relating to the vector and clist variables as necessary. In the present context, recall that during the creation process, the size of the clist variable, TRAJ, was specified as "user-assignable". Hence, the user is prompted for the active size of TRAJ, which is four (Figure 4.29).

At this point, the user has entered all the necessary data required for the execution of the DIFPAK/EX1 combination. NEMS organizes the data as records in a file called NEMS.TXT. The user is prompted by NEMS to perform the file transfer to the mainframe (Figure 4.30).

In the sample environment, the file transfer program and the mainframe computer are respectively the "KERMIT" program and the Amdahl mainframe of the University of Ottawa computing center. Access

```
==============EXECUTE============================================================
    PROBLEM INSTANCE : EX1

                                                DEFAULT          DESIRED
    PARAMETER           TYPE                    VALUES           VALUES
    ---------           ----                    -------          -------

    N                   INTEGER NUMERIC         3                3
    FINTIM              REAL NUMERIC            0.20000D1        1.0
```

Figure 4.27

```
==============EXECUTE============================================================
    PROBLEM INSTANCE : EX1
    PARAMETER
        NAME : XZ
        TYPE : VECTOR NUMERIC REAL
        MAX LENGTH : 10       ACTIVE LENGTH :   3

    COMPONENT           DEFAULT VALUES          DESIRED VALUES
    ---------           --------------          --------------
        1                     0.30000D1           0.30000D1
        2                     0.00000D0           4
        3                     0.00000D0           0.00000D0
```

Figure 4.28

```
==============EXECUTE==========================================================
    PROBLEM INSTANCE : EX1
    VARIABLE
        NAME : TRAJ
        TYPE : CLIST NUMERIC REAL


              ASSIGN SIZE OF CLUSTER LIST :  4
```

Figure 4.29

```
==============EXECUTE==========================================================
    READY FOR FILE TRANSFER...


              INITIATE NEMS-REMOTE BY TRANSFERRING THE FILE

                        NEMS.TXT

                    TO THE AMDAHL COMPUTER.


                        PRESS RETURN TO CONTINUE...
```

Figure 4.30

is via the sytek local area network. NEMS initiates KERMIT for the user. However, the user must explicitly perform the file transfer and activate NEMS-remote himself.

The commands that must be entered to carry out the experiment and to retrieve its results, are as follows :

(a) C(ONNECT) followed by CALL 3301 : this connects the PC to the Amdahl computer.

(b) VT100 : this is entered in response to the query regarding terminal type.

(c) Normal sign-on to Amdahl.

(d) MICRO followed by KERMIT : this activates KERMIT-remote on the Amdahl.

(e) RECEIVE : this KERMIT command prepares Amdahl for receiving a file from the PC.

(f) CTRL-]-C : this returns the user to the PC.

(g) SEND NEMS.TXT : this KERMIT command sends the file NEMS.TXT from the PC to Amdahl. (The file NEMS.TXT is shown in Figure 4.31.)

(h) C(ONNECT) : this once again connects the PC to Amdahl.

(i) NEMS : this command activates NEMS-remote, which separates NEMS.TXT into three files and then activates the selected SE/PI

combination; i.e. carries out the experiment. (The Exec file for NEMS-remote is NEMS EXEC which is shown in Figure 4.32. The activation of the selected SE/PI combination is via the Exec called DIFNEMS which is shown in Figure 4.33.)

(j) KERMIT : this re-activates KERMIT-remote on the Amdahl.

(k) SEND NEMS OUTPUT : this KERMIT command sends the file NEMS OUTPUT from Amdahl to the PC.

(l) CTRL-]-C : this returns the user to the PC.

(m) RECEIVE : this KERMIT command enables the PC is to receive the file NEMS OUTPUT from Amdahl.

(n) C(ONNECT) : this once again connects the PC to Amdahl.

(o) normal sign-off from Amdahl.

(p) CTRL-]-C : this returns the user to the PC.

The file NEMS OUTPUT that is transferred from the Amdahl to the PC contains the values for the variables of both the SE and PI which result from the execution of the experiment. Upon return to the PC (step(p)), these values are placed by NEMS into the appropriate tuples of the appropriate relations. NEMS then returns to the initial state of the main menu.

```
&TRACE OFF
COPY NEMS TXT A DIFPAK INPUT A (FR 16 FOR 2
COPY NEMS TXT A EX1 INPUT A FR 18 FOR 3
EXEC DIFNEMS EX1
ERASE DIFNEMS TXT A
ERASE DIFPAK INPUT A
ERASE EX1 INPUT A
*
*
*
COPY EX1 NCLIST A EX1 OUTPUT A (APPEND
RENAME EX1 CLIST A PI010001 CLIST A
ERASE EX1 NCLIST A
COPY EX1 OUTPUT A DIFPAK OUTPUT A (APPEND
RENAME DIFPAK OUTPUT A NEMS OUTPUT A
          7              0              0.00001D0          0.00001D0
   0.10000D0
          3         0.10000D1           0.30000D1          0.40000D1
   0.00000D0         0.00000D0           0.00000D0          0.00000D0
   0.00000D0         0.00000D0           0.00000D0          0.00000D0
```

Figure 4.31

Contents of the File "NEMS.TXT" for the Example Problem.

```
&TRACE OFF
COPY NEMS TXT A SE EXEC A (FR 1 FOR 15
EXEC SE
ERASE SE EXEC A
```

Figure 4.32

Contents of the File "NEMS EXEC".

```
*CONTROL OFF NOMSG
&FT = FORTX
STATE &1 &FT D
&IF &RETCODE EQ 0 &GOTO -CONT
&FT = FORTRAN
STATE &1 &FT D
&IF &RETCODE EQ 0 &GOTO -CONT
&TYPE FILE (&1 FORTX) OR (&1 FORTRAN) NOT ON YOUR D DISK
&TYPE REQUEST ABORTED
&EXIT
-CONT
&STACK HT
COPY &1 &FT D = = A (REPL
&STACK RT
GLOBAL TXTLIB VLNKMLIB FORTUTIL CMSLIB
GLOBAL TXTLIB DIFPC1 DIFPC2 DISCO TRANLIB VFORTLIB IMSLDLIB
GLOBAL LOADLIB VFLODLIB
EXEC CLEAN
&CONTROL CMS
&SPACE 2
LOAD TRANSL (NOMAP
START TRANSL (&1
&IF &RETCODE NE 0 &EXIT
ERASE &1 &FT A
&SPACE
COPY DIFNEMS INFACE A TARGET FORTRAN A (APPEND
FORTVS TARGET (LANGLVL (66)
&IF &RETCODE LE 4 &GOTO -NOERR
&ERR = Y
&TYPE RETURN CODE FROM FORTVS COMPILER GT 4
&GOTO -XYZ
```

Figure 4.33

Contents of the File "DIFNEMS EXEC".

```
-NOERR
&ERR = N
-XYZ
ERASE TARGET FORTRAN
&SPACE
&IF &ERR EQ Y COPY TARGET LISTING A SOURCE FORTXBL A (AP
ERASE TARGET LISTING A
&IF &ERR EQ Y &GOTO -PRINT
LOAD TARGET (NODUP
&SPACE
FILEDEF 6 DISK &1 STDOUT A
FILEDEF 5 TERM
FILEDEF 12 TERM (LRECL 132
FILEDEF 15 DISK DIFPAK INPUT A
FILEDEF 16 DISK &1 INPUT A
FILEDEF 25 DISK DIFPAK OUTPUT A
FILEDEF 26 DISK &1 OUTPUT A
FILEDEF 35 DISK DIFPAK CLIST A
FILEDEF 36 DISK &1 CLIST A
FILEDEF 45 DISK DIFPAK NCLIST A
FILEDEF 46 DISK &1 NCLIST A
&TYPE READY TO GO ... HIT RETURN TO STRAT
&READ VARS &ANS
EXECSERV CLEARSCN
START
COPY &1 STDOUT A SOURCE FORTXBL A (AP
ERASE LOAD MAP
&SPACE
&TYPE DO YOU WANT TO SEE THE OUTPUT FILE?
&TYPE (<CR> = YES; ANYTHING ELSE = NO)
&READ VARS &RESP
```

Figure 4.33 (Cont'd)

```
&IF .&RESP EQ . TYPE &1 STDOUT A
&STACK RT
&TYPE DO YOU WANT TO ERASE THE OUTPUT FILE?
&TYPE (<CR> = YES; ANYTHING ELSE = NO)
&READ VARS &RESP
&IF &RESP EQ . ERASE &1 STDOUT A
-PRINT
ERASE TARGET TEXT A
&SPACE
RENAME SOURCE FORTXBL A DIFNEMS &1 A
&TYPE DO YOU WANT TO PRINT THE (SOURCE + OUTPUT) FILE?
&TYPE (<CR> = YES; ANYTHING ELSE = NO)
&READ VARS &PR
&IF .&PR NE . &GOTO -NOPRT
EXEC PRINT DIFNEMS &1 A
-NOPRT
&SPACE
ERASE DIFNEMS &1 A
&EXIT
```

Figure 4.33 (Cont'd)

4.6 ILLUSTRATION OF THE QUERY MODE

In the discussion that follows, we assume that the NEMS database contains the following SE's, PI's and experiments :

(a) two SE's called DIFPAK and OPTPAK;

(b) three PI's called EX1, EX2 and EX3 which are bound to DIFPAK and one PI called ROSEN which is bound to OPTPAK;

(c) five experiments : three experiments carried out with DIFPAK and EX1, one experiment with DIFPAK and EX2, and one experiment with OPTPAK and ROSEN (no experiment has yet been carried out with EX3).

To query the NEMS database, the user selects the option "QUERY SE/PI" from the main menu (Figure 4.1) which results in the display of the SHOW/DESIGNATE menu (Figure 4.34). To obtain a list of all the SE's in the NEMS database, the user first enters "1" in the CHOICE prompt field and then enters "?" in the SE prompt field (Figure 4.35). NEMS then displays the list of all the SE's that are known to it (Figure 4.36).

To view the defining attribute values of a particular SE, for example, DIFPAK, the user enters

"DIFPAK:A"

in the SE prompt field (Figure 4.37). This results in the display shown in Figure 4.38.

```
==============QUERY=============================================================
                    SHOW/DESIGNATE MENU
                    -------------------

                    DESIGNATED SE : *NONE*
                    DESIGNATED PI : *NONE*

                                              (NAS = 5)

                         1. SHOW/DESIGNATE        2. REFINE MENU
                            SE OR PI              3. DISPLAY MENU
                                                  4. CLEAR
                                                  5. EXIT TO MAIN MENU
                    CHOICE :  1
================================================================================
================================================================================
```

Figure 4.34

```
==============QUERY=============================================================
                    SHOW/DESIGNATE
                    --------------

                    DESIGNATED SE : *NONE*
                    DESIGNATED PI : *NONE*

                                              (NAS = 5)

                    SE :  ?
                    PI :
================================================================================
================================================================================
```

Figure 4.35

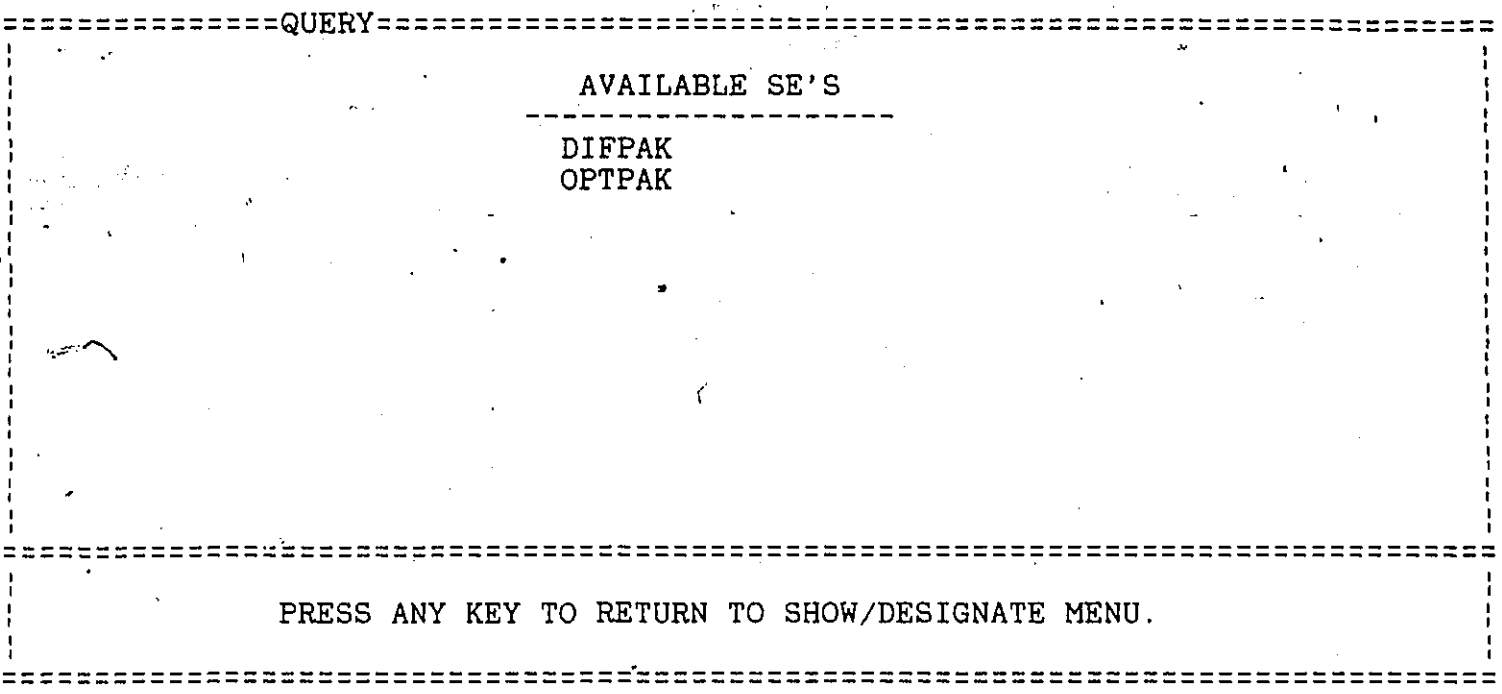


Figure 4.36

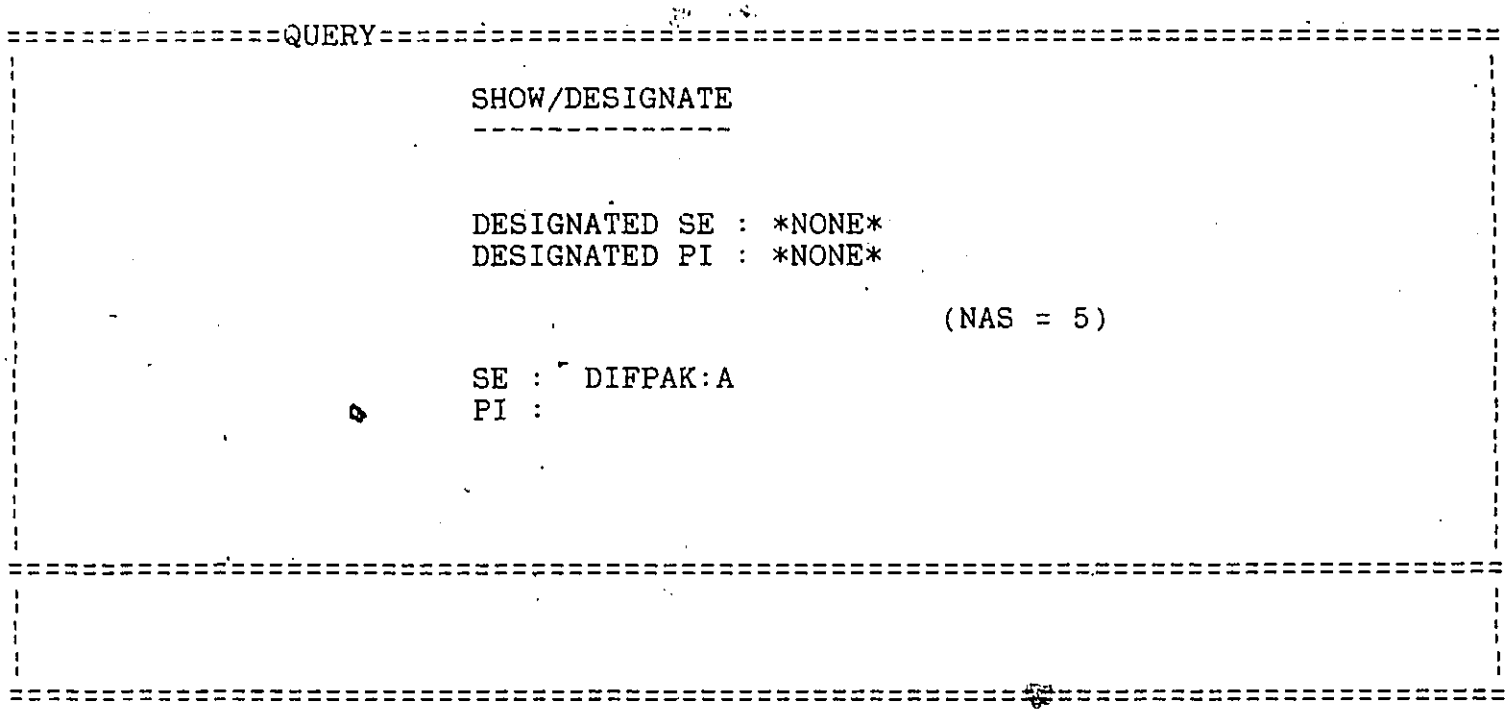


Figure 4.37

```
==============QUERY=============================================================

      SE            INTERNAL  CREATION
     NAME             ID       DATE          OWNER
    ------------------------------------------------
    DIFPAK              SE01  06/26/87    MARIA

     NO OF        NO OF        NO OF        NO OF
    BASIC PAR  REFLECTED PAR  BASIC VAR  REFLECTED VAR
    ------------------------------------------------
        5            3            2            1

                                NO OF
                  PRECISION     USAGES
                  ----------------------
                  DOUBLE          4

================================================================================

          PRESS ANY KEY TO RETURN TO SHOW/DESIGNATE MENU.

================================================================================
```

Figure 4.38

```
==============QUERY=============================================================

                  SHOW/DESIGNATE
                  --------------


                  DESIGNATED SE : *NONE*
                  DESIGNATED PI : *NONE*

                                                  (NAS = 5)

                  SE :  DIFPAK:P
                  PI :

================================================================================

================================================================================
```

Figure 4.39

To view the parameter specifications of DIFPAK, the user enters

"DIFPAK:P"

in the SE prompt field (Figure 4.39) which results in the display of the specifications of the parameters, and reflected parameters of DIFPAK (Figure 4.40 and Figure 4.41). To view the variable specifications of DIFPAK, the user enters

"DIFPAK:V"

in the SE prompt field (Figure 4.42) which results in the display of the specifications of the variables and reflected variables of DIFPAK as shown in Figure 4.43 and Figure 4.44 respectively.

Similarly, to obtain a list of all the PI's known to NEMS, the user enters "?" in the PI prompt field. If, on the other hand, the user wishes to view the specifications of a PI; e.g. EX1, then he enters "EX1:A" to obtain the defining attributes, or "EX1:P" to obtain the parameter specifications, or "EX1:V" to obtain the variable specifications.

The designation of an SE is achieved by simply entering the name of the SE in the SE prompt field (Figure 4.45). This designation (of DIFPAK) results in the display of the SHOW/DESIGNATE menu (Figure 4.34). To view the experiments related to DIFPAK (without any refinement), the "DISPLAY MENU" option is selected from the SHOW/DESIGNATE menu.

```
==============QUERY=============================================================
SOLUTION ENGINE : DIFPAK
BASIC PAR                LOWER              UPPER              DEFAULT
  NAME          TYPE     BOUND              BOUND              VALUES
-------------------------------------------------------------------------------
METHOD          S/INT    0                  100                9
VERIFY          S/INT    0                  1                  0
ABSERR          S/REAL   0.00000D0          0.10000D1          0.00001D0
RELERR          S/REAL   0.00000D0          0.10000D1          0.00001D0
HSTRT           S/REAL   0.00000D0          0.10000D2          0.10000D0

===============================================================================

                PRESS ANY KEY TO GO TO REFLECTED PARAMETER.

===============================================================================
```

Figure 4.40

```
==============QUERY=============================================================
    SOLUTION ENGINE : DIFPAK
          REFL. PAR                MAX          ACTIVE
            NAME          TYPE     LEN          LEN
        -------------------------------------------------------
        N                 S/INT
        FINTIM            S/REAL
        XZ                V/REAL   10             VIA N

===============================================================================

            PRESS ANY KEY TO RETURN TO SHOW/DESIGNATE MENU.

===============================================================================
```

Figure 4.41

```
===============QUERY=================================================
|                         SHOW/DESIGNATE                            |
|                         --------------                            |
|                                                                   |
|                         DESIGNATED SE : *NONE*                    |
|                         DESIGNATED PI : *NONE*                    |
|                                                   (NAS = 5)       |
|                                                                   |
|                         SE :   DIFPAK:V                           |
|                         PI :                                      |
|                                                                   |
=====================================================================
|                                                                   |
=====================================================================
```

Figure 4.42

```
==============QUERY==================================================
|    SOLUTION ENGINE : DIFPAK                                       |
|          BASIC VAR                MAX          ACTIVE             |
|            NAME           TYPE    LEN/SIZE     LEN/SIZE           |
|         ---------------------------------------------------       |
|         NRHS             S/INT                                    |
|         AVG              S/REAL                                   |
|                                                                   |
=====================================================================
|                                                                   |
|            PRESS ANY KEY TO GO TO REFLECTED VARIABLE.             |
|                                                                   |
=====================================================================
```

Figure 4.43

```
==============QUERY=============================================================
  SOLUTION ENGINE : DIFPAK
        REFL. VAR                    MAX          ACTIVE
        NAME            TYPE         LEN/SIZE     LEN/SIZE
        ------------------------------------------------------------
        TRAJ            C/REAL       N/A          USER ASSIGNABLE
```

```
================================================================================
            PRESS ANY KEY TO RETURN TO SHOW/DESIGNATE MENU.
================================================================================
```

Figure 4.44

```
==============QUERY=============================================================
                      SHOW/DESIGNATE
                      --------------


                      DESIGNATED SE : *NONE*
                      DESIGNATED PI : *NONE*

                                                (NAS = 5)

                      SE :  DIFPAK
                      PI :
```

```
================================================================================
```

Figure 4.45

Within the DISPLAY menu (Figure 4.46), the choice of the "PI's" option provides a list of all the PI's that are bound to the designated SE; namely, DIFPAK (Figure 4.47). The P/SE option of the DISPLAY menu provides the means for viewing the values of the parameters of the designated SE in the various experiments carried out with it. This display is carried over the set of experiments currently in the active set.

The experiments are processed in an order governed by the following two rules :

(a) all experiments associated with a particular PI appear together in a chronological sequence;

(b) the order in which the PI's are selected is also chronological, based on the first experiment with each PI.

Not all experiments, however, need to be displayed. After the display of the SE parameter values for any particular experiment (Figure 4.48), the user is able to specify a number of subsequent experiments that are to be skipped (Figure 4.49).

The procedure for displaying the values of variables in the experiments involving the currently designated SE is entirely analogous. This procedure would be activated by selecting the V/SE option from the DISPLAY menu.

```
==============QUERY=============================================================

                        DISPLAY MENU
                        ------------

                DESIGNATED SE : DIFPAK
                DESIGNATED PI : *NONE*
                                                (NAS = 4)

                     1. PIs            6. SHOW/DESIG MENU
                     2. P/SE           7. REFINE MENU
                     3. P/PI           8. CLEAR
                     4. V/SE           9. EXIT TO MAIN MENU
                     5. V/PI
                CHOICE :  1
```

Figure 4.46

```
==============QUERY=============================================================
  DESIGNATED SE : DIFPAK
  DESIGNATED PI : *NONE*

                              ASSOCIATED PI'S
                            --------------------
                              EX1
                              EX2
                              EX3

================================================================================

                  PRESS ANY KEY TO RETURN TO DISPLAY MENU.

================================================================================
```

Figure 4.47

```
==============QUERY=============================================================
  SOLUTION ENGINE   : DIFPAK              NO OF EXPT REMAINING = 3
  PROBLEM INSTANCE  : EX1
  EXPERIMENT DATE   : 07/06/87

        PAR NAME            TYPE          ACTIVE VALUES
        ------------------------------------------------
        METHOD              S/INT         9
        VERIFY              S/INT         0
        ABSERR              S/REAL        0.00001D0
        RELERR              S/REAL        0.00001D0
        HSTRT               S/REAL        0.10000D0

================================================================================

                    RETURN TO DISPLAY MENU? (Y/N)

================================================================================
```

Figure 4.48

```
==============QUERY=============================================================
  SOLUTION ENGINE   : DIFPAK              NO OF EXPT REMAINING = 3
  PROBLEM INSTANCE  : EX1
  EXPERIMENT DATE   : 07/06/87

        PAR NAME            TYPE          ACTIVE VALUES
        ------------------------------------------------
        METHOD              S/INT         9
        VERIFY              S/INT         0
        ABSERR              S/REAL        0.00001D0
        RELERR              S/REAL        0.00001D0
        HSTRT               S/REAL        0.10000D0

================================================================================

          ENTER THE NO OF EXPT TO BE SKIPPED (IF ANY). 1

================================================================================
```

Figure 4.49

Instead of displaying, for each experiment, the parameters associated with the designated SE, the user may wish to display the parameter values of the PI. This can be achieved simply by selecting the P/PI option in the DISPLAY menu. Similarly, the variables of the PI can be displayed by selecting the V/PI option.

Suppose now that the user decides to focus his interest on the experiments that are associated with the problem instance EX1. To achieve this goal, the query subsystem must first be reset by selecting the "CLEAR" option from the DISPLAY menu (Figure 4.50). This results in the display of the SHOW/DESIGNATE menu (Figure 4.34). The user first enters "1" in the CHOICE field and then enters "EX1" in the PI prompt field, thereby designating EX1 (Figure 4.51). (Note that since DIFPAK is the SE_master of EX1, DIFPAK is implicitly designated.) The designation of EX1 results in the display of the SHOW/DESIGNATE menu (Figure 4.34).

Suppose now that the user is not interested in displaying the value of the DIFPAK parameter ABSERR. This masking operation is implemented by first selecting the option "MASK" from the REFINE menu (Figure 4.52). This results in the display of the menu as shown in Figure 4.53. Since ABSERR is an SE parameter, the user enters 1 in the CHOICE prompt field. NEMS then prompts the user for the name of the SE

```
==============QUERY=============================================================

                        DISPLAY MENU
                        ------------

                   DESIGNATED SE : DIFPAK
                   DESIGNATED PI : *NONE*
                                                (NAS = 4)

                        1. PIs          6. SHOW/DESIG MENU
                        2. P/SE         7. REFINE MENU
                        3. P/PI         8. CLEAR
                        4. V/SE         9. EXIT TO MAIN MENU
                        5. V/PI
                   CHOICE :  8

================================================================================

================================================================================
```

Figure 4.50

```
==============QUERY=============================================================

                   SHOW/DESIGNATE
                   --------------


                   DESIGNATED SE : *NONE*
                   DESIGNATED PI : *NONE*

                                                (NAS = 5)

                   SE :
                   PI :  EX1


================================================================================

================================================================================
```

Figure 4.51

```
==============QUERY=============================================================
                          REFINE MENU
                          -----------

                    DESIGNATED SE : DIFPAK
                    DESIGNATED PI : EX1
                                                  (NAS = 3)

                        1. SPECIALIZE   5. SHOW/DESIG MENU
                        2. RESTORE      6. DISPLAY MENU
                        3. MASK         7. CLEAR
                        4. UNMASK       8. EXIT TO MAIN MENU
                    CHOICE :  3
```

Figure 4.52

```
==============QUERY=============================================================
  DESIGNATED SE : DIFPAK
  DESIGNATED PI : EX1

                    MASK
                         1. SE PARAMETER
                         2. SE VARIABLE
                         3. PI PARAMETER
                         4. PI VARIABLE
                    CHOICE :  1
```

```
                    ENTER A CHOICE OR PRESS RETURN TO REFINE MENU...
```

Figure 4.53

parameter to be masked (Figure 4.54).

Suppose, in addition, that the user is interested only in those experiments where the value of the DIFPAK variable NRHS is greater than 200 and less than or equal to 500. This is a specialization operation and is initiated with the selection of the "SPECIALIZATION" option from the REFINE menu (Figure 4.52). This results in the display of a menu as shown in Figure 4.55.. Since NRHS is a DIFPAK variable, the user enters "2" in the CHOICE prompt field. NEMS then prompts the user for the name of the SE variable to be specialized, and the user enters "NRHS" (Figure 4.56). This results in the display as shown in Figure 4.57. Via this display, the user is required to enter a specific value or a value-range for the DIFPAK variable NRHS.

Having thus carried out masking and specialization operations, the user can return to the DISPLAY menu to make appropriate selections. The parameter/variable displays that are subsequently presented will be subject to the masking and specialization operations that are in effect.

To unmask the DIFPAK parameter, ABSERR, the user first selects the "UNMASK" option from the REFINE menu and then, on the subsequent display of Figure 4.58, he chooses the "SE PARAMETER" option. Note that this unmasks all masked parameters of the currently designated SE;

```
==============QUERY=============================================================
  DESIGNATED SE : DIFPAK
  DESIGNATED PI : EX1


                  MASK
                       1. SE PARAMETER
                       2. SE VARIABLE
                       3. PI PARAMETER
                       4. PI VARIABLE
                  CHOICE :  1 :

                  ENTER NAME OF SE PARAMETER :  ABSERR
================================================================================

================================================================================
```

Figure 4.54

```
==============QUERY=============================================================
  DESIGNATED SE : DIFPAK
  DESIGNATED PI : EX1


                  SPECIALIZE
                       1. SE PARAMETER
                       2. SE VARIABLE
                       3. PI PARAMETER
                       4. PI VARIABLE
                  CHOICE :  2




================================================================================
                  ENTER A CHOICE OR PRESS RETURN TO REFINE MENU...
================================================================================
```

Figure 4.55

```
==============QUERY=============================================================
DESIGNATED SE : DIFPAK
DESIGNATED PI : EX1

                SPECIALIZE
                     1. SE PARAMETER
                     2. SE VARIABLE
                     3. PI PARAMETER
                     4. PI VARIABLE
                CHOICE :  2

                ENTER NAME OF SE VARIABLE :  NRHS
================================================================================

================================================================================
```

Figure 4.56

```
==============QUERY=============================================================
DESIGNATED SE : DIFPAK
DESIGNATED PI : EX1

                SPECIALIZE SE VARIABLE : NRHS

                ENTER RANGE INFORMATION :

                     =
                     >   200
                     <
                     >=
                     <=  500

================================================================================

================================================================================
```

Figure 4.57

```
==============QUERY=============================================================
  DESIGNATED SE : DIFPAK
  DESIGNATED PI : EX1

                    UNMASK
                         1. SE PARAMETER
                         2. SE VARIABLE
                         3. PI PARAMETER
                         4. PI VARIABLE
                    CHOICE :  1

================================================================================

                    ENTER A CHOICE OR PRESS RETURN TO EXIT...

================================================================================
```

Figure 4.58

NEMS does not provide the means for selectively unmasking parameters or variables.

The resetting of the existing specialization (i.e. on NRHS) proceeds in a similar way. In particular, the RESTORE option is selected from the REFINE menu.

The preceding discussion has outlined the activation and deactivation of refinements (i.e., mask, unmask, specialize, restore) in the context of an SE. Entirely analogous steps are followed for carrying out this operation in the context of a PI.

When the user finishes querying the NEMS database, he can exit the query subsystem by selecting the option "EXIT TO MAIN MENU". This results in the display of the NEMS main menu from which the user can carry out other NEMS activities or exit from the system.

4.7 ILLUSTRATION OF THE DELETE MODE

Suppose the user wishes to delete the solution engine OPTPAK from the NEMS database. The user selects the option "DELETE SE/PI" from the main menu (Figure 4.1). This results in the display of the DELETE menu (Figure 4.59), from which the "DELETE SE" option is selected. NEMS then prompts the user for the name of the SE that is to be deleted (Figure

4.60) and proceeds to carry out the steps as outlined in Section 3.4. The procedure for deleting a PI is entirely analogous.

After a deletion operation is completed, NEMS re-displays the main menu thereby allowing the user to activate any of the four NEMS modes or to exit NEMS.

```
===============DELETE==================================================
|                                                                     |
|                     DELETE MENU                                     |
|                          1. DELETE SE                               |
|                          2. DELETE PI                               |
|                     CHOICE :  1                                     |
|                                                                     |
=======================================================================
|                                                                     |
=======================================================================
```

Figure 4.59

```
===============DELETE==================================================
|                                                                     |
|                     DELETE MENU                                     |
|                          1. DELETE SE                               |
|                          2. DELETE PI                               |
|                     CHOICE :  1                                     |
|                                                                     |
|                     ENTER NAME OF SE TO BE DELETED :  OPTPAK        |
|                                                                     |
=======================================================================
|                                                                     |
=======================================================================
```

Figure 4.60

CHAPTER 5

CONCLUSIONS

5.1 CONCLUDING REMARKS

In this thesis project, we have designed and developed the Numerical Experiments Management System (NEMS) which provides a productivity tool for researchers and designers who have the need to carry out extensive sets of numerical experiments. The basic purpose of the tool is to automate the management of the experiments and the results which are generated by these experiments. A means for browsing through the accumulated experimental results is also provided. This latter feature is intended to provide a convenient means for assessing the results and obtaining insight into further experiments that need to be carried out.

The model of the problem solving environment upon which the NEMS development is based, consists of three basic entities; namely, solution engines, problem instances and experiments. Each of these is characterized by a set of attributes and values for these attributes are maintained in a dBASE III database. The attributes have been carefully chosen so that a reasonably broad class of numerical experimentation

activity can be accommodated by the system. We have described the basic entities and outlined the design of the relations which are used to manage the data associated with the experimentation.

We have also described the four NEMS functional modes; namely, (a) the create mode which allows the user to create instances of solution engines and problem instances; (b) the execute mode which provides the means for executing a compatible solution engine/problem instance combination thereby creating an instance of an experiment object; (c) the query mode which provides a convenient means for browsing through experimental results; and (d) the delete mode which allows the user to delete an instance of a solution engine or a problem instance.

The software and hardware environments upon which NEMS is based, and the various user requirements in order that the user can use NEMS efficiently, have also been outlined. We have also presented the underlying dBASE III relations that are used in NEMS and have discussed the structure and function of each relation in detail. In addition, we have described some of the pertinent implementational aspects of the four NEMS modes and have presented a typical example which illustrates the use of NEMS and, hopefully, will also serve as a user's guide.

NEMS has been implemented using dBASE III and some difficulties

arising from this choice were encountered during the development process. As outlined in Section 2.2.2, one such problem with dBASE III lies in its inadequate way of handling numeric data (e.g. scientific notation is not supported). To deal with this difficulty, NEMS branches out to TURBO PASCAL. However, in TURBO PASCAL, the smallest and largest integers that can be handled are -32768 and 32767, respectively, and this imposes a restriction on the range of integer value that a user can use.

A question may arise as to why we did not implement the whole system using an algorithmic language such as TURBO PASCAL in the first place. The main reason is simply to have access to the powerful means provided by dBASE III for modifying and querying a database. If the user wishes to modify the basic structures of the relations in NEMS, he can do so in a very simple and straightforward way, with only minor changes in the software. Furthermore we note that if the user has a good knowledge of dBASE III and the internal structures of the NEMS database, he can query the NEMS database directly through the dBASE III facilities without going through the query subsystem provided by NEMS. This can be particularly useful for extracting information from the database which is not accessible from the NEMS query subsystem.

As we have indicated in Section 3.1.2, a collection of dBASE III relations is maintained which serves as the templates for the creation of new relations associated with each instance of a solution engine or problem instance that is created in NEMS. In its present form, NEMS can accommodate a maximum of fifteen parameters and fifteen variables for each solution engine and problem instance, and each vector parameter or vector variable can have a maximum of ten components. These are arbitrary choices and if the user finds these sizes to be inadequate, he can increase them with minor changes to the collection of NEMS template relations.

5.2 FUTURE WORK

In the present version of NEMS, a problem instance is bound to a unique solution engine. This could prove to be unreasonable restriction and it might be desirable to make changes to NEMS to allow greater flexibility in this regard. These changes could be implemented relatively easily.

As currently designed, the data values of a clist variable are stored in a file which is resident on the mainframe. It would most certainly be desirable to incorporate a graphics capability in the NEMS software which

would allow the convenient retrieval of the data values of a clist and their presentation in a graphical form on the PC screen. It might also be useful to remove the current restriction of a single clist variable for any particular solution engine or problem instance.

The incorporation of a graphical display feature could be regarded as a first step in the direction of post-processing capabilities. The general purpose of such developments would be to make possible a range of analyses to be carried out over the accumulated experimental data. An enhanced NEMS environment with these features would substantially extend the range of inferences that could be made from the accumulated data. The realization of such a goal would represent a significant research project.

In the create mode, we have provided the user with four options for specifying the active length of a vector; namely, "fixed", "user-assignable", "via the value of some scalar parameter", or "via the active length of another vector". Similarly, the active size of a clist can be "fixed", "user-assignable", or "via the value of some scalar parameter". There are, however, situations where the active length of a vector or the active size of a clist needs to be specified as a function of some scalar parameter. This, in fact, occurred in the example treated in Chapter Four where the

size of the clist variable TRAJ is (N+1). The current implementation of NEMS cannot accommodate a length/size specification which is expressed as a function and such an extension would be worthwhile.

Perhaps the most ambitious possible extension of NEMS would be an attempt to extend its role from a passive support tool to an active support tool by incorporating goal directed features. For example in any particular installation, NEMS could provide access to a large variety of numerical software packages (i.e. solution engines). A valuable addition therefore would be an advisor subsystem that would provide guidance to the user in choosing the solution engine that most closely matches his specific problem, or where applicable, providing an explanation of the pro's and con's of several equally applicable engines.

Our experience with NEMS has demonstrated its capability as an effective experiment management system. It will undoubtedly evolve and this evolution will be driven, for the most part, by user feedback. Hopefully the current implementation has sufficient modularity and structural flexibility to accommodate these evolutionary changes.

REFERENCES

[1] T.I. Ören and B. Zeigler, "Concepts for Advanced Simulation Methodologies", Simulation, vol. 32, no. 3, March, 1979, pp. 69-82.

[2] C.R. Standridge, "Using the Simulation Data Language (SDL)", Simulation, vol. 37, no. 3, September, 1981, pp. 73-81.

[3] C.R. Standridge, "The Simulation Data Language (SDL) : Applications and Examples", Simulation, vol. 37, no. 4, October, 1981, pp. 119-130.

[4] G.H. Korn, "A New Interactive Environment for Computer-Aided Experiments", Simulation, vol. 45, no. 6, December, 1985, pp. 303-305.

[5] L.G. Birta, "DIFPAK : A Software Package for Solving ODE's and Evaluating ODE Codes", Report TR-85-06, Department of Computer Science, University of Ottawa, January, 1985.

[6] L.G. Birta, "A Quasi-Parallel Method for the Simulation of Loosely Coupled Continuous Subsystems", Mathematics and Computers in Simulation, vol. 22, 1980, pp. 189-199.

[7] L.G. Birta, "The Trade-off Between Accuracy and Overhead in Continuous Subsystems", Proc. 1981 Summer Computer Simulation Conference, Washington, D.C. July 15-17, 1981, pp. 88-95.

[8] L.G. Birta, T.I. Ören and D.H. Kettenis, "A Robust Procedure for Discontinuity Handling in Continuous System Simulation", Trans. of the Society for Computer Simulation, vol. 2, no. 3, September, 1985, pp. 189-205.

[9] L.G. Birta and O. Abou-Rabia, "Parallel Block Predictor-Corrector Methods for ODE's", IEEE Trans. on Computers, vol. c-36, no. 3, March, 1987, pp. 299-311.

[10] L.G. Birta and M. So, "A Database System for the Management of Numerical Experiments", Proc. Summer Computer Simulation Conference, Reno, Nevada, July 28-30, 1986, pp. 38-43.

[11] C.J Date, An Introduction to Database Systems, vol. 1. Addison-Wesley, 1982.

[12] C.J Date, An Introduction to Database Systems, vol. 2. Addison-Wesley, 1983.

[13] L. Castro, J. Hanson and T. Rettig, Advanced Programmer's Guide Featuring dBASE III and dBASE II. Ashton-Tate, 1985.