This chapter discusses the design and implementation details of Neuro-Expert (Figure 1).

Figure 1. Neuro-Expert block diagram.
In this chapter, we set the stage with an insight into the system architecture and higher level modules. In the first section we look at the system from the designer's perspective. We shall dissect the system and take an in-depth look at each component of the system. This section also presents the main block-diagram and data flow diagram. We will briefly discuss the major modules of our system and explain how mapping facility has been developed to map rules from knowledge base to an Artificial Neural Network.

In the second section we look at the system from programmer's perspective. We touch upon the syntactical interfaces and the functions provided by Knowledge Definition Language (KDL). The section ends with sample rules from the Shares Knowledge base. This knowledge base has more than 200 rules. We are presenting here sample rules to indicate our knowledge engineering methodology.

In the third section, we take a look at the system from a user's perspective. Issues included here are, how easy it is to use and learn the system. This section describes the user interface in detail.

6.1 System Design

The system (Figure 2) can be decomposed into the following major modules (Figure 3):

- Lexical Analyzer
- Inference Engine
- Interpreter
- Database Interface
- Knowledge Base
- User Interface
- NN Interface
- NN Training

Figure 2. Level-0 data flow diagram for Neuro-Expert.
Figure 3. Functional decomposition diagram for Neuro-Expert.
6.1.1 Lexical Analyzer

The lexical analyzer (Figure 4) is an important module of the system that loads the knowledge base into memory and analyses it for both syntactic and semantic errors. This module refines the knowledge base by removing correct lines and unused blank lines. The lexical analyzer has been developed after a careful design of the language structure.

Figure 4. Data flow diagram for Lexical-Analyzer.
6.1.2 Inference Engine

The Inference Engine (Figure 5) consists of 3 algorithms for match-rules, select-rules and execute-rules and directing supports forward chaining with match rules matching the condition elements against invoking memory, select rules choosing one dominant rule and execute-rules firing the rules by executing the actions of Right Hand Side (RHS) sequence while the inference engine is a forward chaining engine, backward chaining problem-solving strategies can be implemented.

Figure 5. Block diagram for Inference Engine,
6.1.3 Interpreter

The interpreter is invoked only by the Inference engine. It analyzes statements in sequence and passes the result back to the Inference Engine, which in turn makes use of this result to take decisions and appropriate actions.

6.1.4 Knowledge Base

The Knowledge Base is an ASCII text file. The first step in creating acknowledge base is the acquisition of the expertise of experts from various sources such as text books, journals, heuristics etc. The knowledge thus obtained is to be coded in a form specified by the grammar rules of the Knowledge Definition Language (KDL). The Knowledge Definition Language is discussed in detail in Section 6.2.

6.1.5 Database Interface

Prime importance is given for the response time of the system. As a first step maximum care is taken to avoid asking unnecessary questions to the user. The data required by the system can be kept in a database file.

A database interface (Foxpro) is integrated with the system. The database filename and key field (to be used in locating a record) should be known. After reading a record the field names and their corresponding values will be available in the symbol table, so that later the field names can be considered to be user defined variables.

The design of the system has been kept open. Database interface for other popular RDBMS's like Oracle, Ingres etc., also could be developed and connected to Neuro Expert.
6.1.6 The User Interface

The Shell has an extremely user friendly interface. The various features of user interface include:

(a) Menus: With the user friendly interface provided with shell, it is easy to develop and use expert systems. The pulldown and popup menus enable the user to operate the system without having to refer to any other user guides of the software.

(b) Context-sensitive help: The system provides a context-sensitive help facility. If the help is invoked from the main menu, an index for help will be displayed, from which items can be selected using the cursor control keys. Pressing F5 from any help screen will take you back to the help index. The Up and Down arrow keys can be used to view through the help displayed.

The indicators at the top and bottom corners of the right side of the help window show the availability of more pages. The help can be activated at any point of time (at any menu).

(c) Integrated editor: To invoke the editor, choose Edit from the main menu. The user can edit either a knowledge base file or non-knowledge base file. The types can be selected using the Edit Sub menu.

When editing is over, the main menu will be restored. If the last edited file is a knowledge base file, then it will be automatically loaded into the system work area. At the present form of the system "Norton editor" is used for editing.

136
6.1.7 NN Interface

This module performs the function of mapping the rules in the knowledge base into the corresponding neural net. In the process of reasoning, each premise is taken as a neural node. Certainty factor of a rule is taken as a rule strength. The premises which are not involved in the conclusion part of any rule, in rule base, are considered as input node of a neural network. Premises that are in conclusion part of a rule are not specified in condition part of another rule are taken as output nodes of a neural network. Rest of the premises are taken as intermediate/hidden nodes. Premises whose certainty strengths (bias values) are not known, are taken as zero.

6.1.7.1 Sample file formats

All the Rule strengths and Premise strengths should be within the range of -1.0 & +1.0

6.1.7.1.1 User file format (Rules)

User will be giving the filename which consists of rules. This file would have an extension ".knb". Format of that file should be:

Ex:

```
(#rule 1:
   (BC (0.87)) & (DE (0.78))
       → (G & FG) (0.98))

(#rule 2:
   (~G) | (BC (0.67)) & (FG (0.87))
       → (X) (0.83))
```
6.1.7.1.2 User file format (Desired Evidences)

This file includes the information about the desired evidences. This file would have an extension ".evi". It should include

- No. of evidences
- Each premise and its desired evidence (separated by a blank)

Ex:

3
G 0.9
LI 0.9 9
YG 1.0

6.1.7.1.3 Internal file format I

This is an internal file used by the program to store the modified rule forms of the given rule base in the user file name. This file would have an extension ".dat".

It consists of eight attributes.

(i) Rule Strength : float value
(ii) No. of Conclusions : int value
(iii) Each conclusion (premise) : char string
(iv) Each premise strength : float value
(v) No. of conditions : int value
(vi) List of operators : char array
(vii) Each condition (premise) : char string
(viii) Condition strength : float value

138
"Shares.dat" for the example given above is

0.98 2 G 0 FG 0.67 & BC 0.67 DE 0.78

0.83 1 X 0 3 | & G 0 BC 0.67 FG 0.67

0.88 2 YG 0 E 0 3 & | X 0 DE 0.73 G 0

0.67 1 LI 0 2 | X 0 DF 0.7

6.1.7.1.4 Internal file format II

It consists of the neural net developed for the given rule base which consists of operation (function) followed by parameters. This file will have an extension ".net".

Ex: Neural net developed for the given rule base

```
MIN BC DE
SN 0 G
SN P FG
MAX G BC
MIN + FG
SN 1 X
MIN X DE
MAX + G
SN 2 YG
SN 2 E
MAX X DF
SN 3 LI
```
6.1.7.2 Neural net format

Example is given in the format of "shares.net". Information in the net will indicate the following features: (pl & p2 indicates premise1 and premise2).

1. For each condition part (which consists of more than one premise)
   a) If the operation between first two premises is
      - ' & ' : MIN pl p2 is written in the file
      - ' | ' : MAX pl p2 is written in the file
   b) For the rest of the premises (if they are more than two)
      • If the operation is
        - ' & ' : MIN + pl is written in the file
        - ' | ' : MAX + p2 is written in the file

2) If condition part consists of one premise
   CNE Rule no. pl is written in the file

3) After condition part is over, special node is invoked to calculate modified premise strength for each conclusion. For each conclusional premise, the following statement is written in a file.
   SN Rule no. pl is written in the file
6.1.7.3 Special nodes developed

- **ADD(x₁,x₂)** — Neural net that adds its two inputs \( x₁ \) and \( x₂ \), and outputs that **sum**.

- **SUB(x₁,x₂)** — Neural net that subtracts one input from another and outputs that result.

- **MULT(x₁,x₂)** — Neural net that multiplies its two inputs and outputs that result.

- **DIV(x₁,x₂)** — Neural net that divides one input by another and outputs that result.

- **MIN(x₁,x₂)** — Neural net that outputs the minimum of \( x₁ \) & \( x₂ \).

- **MAX(x₁,x₂)** — Neural net that outputs the maximum of \( x₁ \) & \( x₂ \).

- **POS(x₁)** — Neural net that outputs 1 if \( x₁ \geq 0 \) else it outputs 0.

- **EQU₁(x₁)** — Neural net that outputs 1 if \( x₁ = 1.0 \) else 0.

- **GRE₁(x₁)** — Neural net that outputs 1 if \( x₁ > 1.0 \) else 0.

- **NEGl(x₁)** — Neural net that outputs if \( x₁ < 0 \) else 0.

- **LESl(x₁)** — Neural net that outputs if \( x₁ < 0 \) else 0.

- **BPOS(x₁,x₂)** — Neural net that outputs 1 if \( x₁ > 0 \) and \( x₂ > 0 \) else it outputs 0.
• **BNEG(x1,x2)** — Neural net that outputs 1 if x1 < 0 and x2 < 0 else it outputs 0.

• **OPP(x1,x2)** — Neural net that outputs 1 if x1 * x2 are of opposite sign else it outputs 0.

• **ABS(x1)** — Neural net that outputs the absolute value of x1.

• **CXCFPOS (CS,CF)** — Neural net that outputs **modified** values of X by taking initial value of conclusion (CX) and rule strength (CF) as positive valued inputs by using the below formula.

\[ C(X|A) = C(X) + CF \times [1.0 + C(X)] \]

• **CXCFNEG(CX,CF)** — Neural net that outputs modified values of X by taking initial value of conclusion (CX) and rule strength (CF) as negative valued inputs by using the below formula.

\[ C(X|A) = C(X) + CF \times [1.0 + C(X)] \]

• **CXCFOPP(CX,CF)** — Neural net that outputs modified value of X by taking initial value of X and rule strength (CF) as opposite sign valued inputs.

\[ C(X|A) = \frac{[C(X) + CF]}{[1.0 - \min\{|C(X)|, |CF|\}]} \]

• **NODEA(A,CF,X)** — Special neural net developed to find out the modified premise strength of conclusion X and modified Rule strength CF, by taking initial conditional strength X. This handles certainty paradigm.
6.1.8 NN Training

After propagating belief values through neural net, net will be reaching the final conclusion with some confidence values for final conclusions. If those confidence values are not the desired one, then the net has to be trained accordingly to reach the desired values. Training will be done only for the final conclusion, i.e., for the nodes at output layer. We are assuming the desired values are used to represent the knowledge which we want to embed in the network. The process of training was discussed in Chapter 4. Examples on training are discussed now.

Some Examples:

**Example 1: Trained**

Give the file name which consists the Rule base two.dat

**INPUT TO THE PROGRAM**

No. of rules in the given file = 4
rule [ 0] is 0.9 B & C -> F & G
rule [ 1] is 0.67 C -> D
rule [ 2] is 0.8 D -> F
rule [ 3] is 0.78 B -> C

evidences b = 0.400000

**RULE STRENGHTS**

CFRS[ 0] = 0.900000
CFRS[ 1] = 0.670000
CFRS[ 2] = 0.870000
CFRS[ 3] = 0.780000
PREMISE STRENGTHS

VARI = F  EVIDI = 0.000000  EVID2 = 0.000000
VARI = G  EVIDI = 0.000000  EVID2 = 0.000000
VARI = B  EVIDI = 0.400000  EVID2 = 0.000000
VARI = C  EVIDI = 0.000000  EVID2 = 0.000000
VARI = D  EVIDI = 0.000000  EVID2 = 0.000000
VARI = ~F  EVIDI = 0.000000  EVID2 = 0.000000

NEURAL NET DEVELOPED

ONE  3  B
SN  3  C
MIN  B  C
SN  0  F
SN  0  G
ONE  1  C
SN  1  D
ONE  2  D
SN  2  ~F

After PROPAGATION values are

RULE strengths

CFRS1 = 0.900000  CFRS2 = 0.280800
CFRS1 = 0.670000  CFRS2 = 0.209040
CFRS1 = 0.870000  CFRS2 = 0.181865
CFRS1 = 0.780000  CFRS2 = 0.312000

PREMISE strengths

VAR = F  EVIDI = 0.000000  EVID2 = 0.229732
VAR = G  EVIDI = 0.000000  EVID2 = 0.280800
VAR = B  EVIDI = 0.400000  EVID2 = 0.400000
VAR = C  EVIDI = 0.000000  EVID2 = 0.312000
VAR = D  EVIDI = 0.000000  EVID2 = 0.209040
VAR = ~F  EVIDI = 0.000000  EVID2 = 0.770268
GIVE THE FILENAME WHICH CONSISTS OF DESIRED EVIDENCES
file name = two.evid
Reading desired evidences NS = 1
CAR[ 0] = G NEW[ 0] = 0.900000

SYSTEM IS ABLE TO TRAIN THE NET TO GET THE DESIRED VALUES
VALUES WILL BE SHOWN

FINAL MODIFIED VALUES ARE
VARI[ 0] = F EVID1 = 0.000000 EVID2 = 0.153131
VARI[ 1] = G EVID1 = 0.860957 EVID2 = 0.900000
VARI[ 2] = B EVID1 = 1.000000 EVID2 = 1.000000
VARI[ 3] = C EVID1 = 0.000000 EVID2 = 0.780000
VARI[ 4] = D EVID1 = 0.000000 EVID2 = 0.522600
VARI[ 5] = "F EVID1 = 0.000000 EVID2 = 0.846869

RULES STRENGTHS
I = 0 CFRS1 = 0.360000 CFRS2 = 0.280800
I = 1 CFRS1 = 0.670000 CFRS2 = 0.522600
I = 2 CFRS1 = 0.870000 CFRS2 = 0.454662
I = 3 CFRS1 = 0.780000 CFRS2 = 0.780000

GOOD BYE

Example 2: Untrained

Give the file name which consists of Rule base one.dat

INPUT TO THE PROGRAM
No. of rules in the given file
rule [ 0] is 0.9 E -> A
rule [ 1] is 0.89 E -> A"
rule [ 2] is 0.78 X -> Y
No. of evidences: 2

evidences E = 0.900000
evidences X = 0.560000

RULE STRENGTHS
CRFS[ 0] = 0.900000
CRFS[ 1] = 0.890000
CRFS[ 2] = 0.780000

PREMISE STRENGTHS
VARI = A EVIDI = 0.000000 EVID2 = 0.000000
VARI = E EVIDI = 0.670000 EVID2 = 0.000000
VARI = Y EVIDI = 0.000000 EVID2 = 0.000000
VARI = X EVIDI = 0.560000 EVID2 = 0.000000

NEURAL NET DEVELOPED
ONE 0 E
SN 0 A
ONE 1 E
SN 1 "A
ONE 2 X
SN 2 Y

After PROPAGATION values are

RULE strengths
CRFS1 = 0.900000 ☰ CRFS2 = 0.603000
CRFS1 = 0.890000 ☰ CRFS2 = 0.596300
CRFS1 = 0.780000 ☰ CRFS2 = 0.436800

PREMISE strengths
VAR = A EVIDI = 0.000000 EVID2 = 0.243431
VAR = E EVIDI = 0.670000 EVID2 = 0.670000
VAR = A EVIDI = 0.000000 EVID2 = 0.756569
VAR = Y EVIDI = 0.000000 EVID2 = 0.436800
VAR = X EVIDI = 0.560000 EVID2 = 0.560000

GIVE THE FILENAME WHICH CONSISTS OF DESIRED EVIDENCES

file name = one.evid

Reading desired evidences \!\!S = 2
CAR[ 0] = Y NEW[ 0] = 0.390000
CAR[ 1] = A NEW[ 1] = 0.900000

SYSTEM IS NOT ABLE TO TRAIN THE NEURAL NET TO GIVE THE DESIRED RESULT

INCONSISTENCY !!!

Inconsistency in Rule Base
Conflicting rules may be existing in the given rule base
Please check it
GOOD LUCK FOR THE NEXT TIME !!!

FINAL MODIFIED VALUES ARE
VARI[ 0] = A EVIDI = 0.748111 EVID2 = 0.099000
VARI[ 1] = E EVIDI = 1.000000 EVID2 = 1.000000
VARI[ 2] = "A EVIDI = 0.000000 EVID2 = 0.901000
VARI[ 3] = Y EVIDI = 0.804688 EVID2 = 0.890000
VARI[ 4] = X EVIDI = 0.560000 EVID2 = 0.560000

RULE STRENGTHS
I = 0 CFRS1 = 0.603000 CFRS2 = 0.603000
I = 1 CFRS1 = 0.890000 CFRS2 = 0.890000
I = 2 CFRS1 = 0.780000 CFRS2 = 0.436800

GOOD BYE
6.2 Knowledge Representation

This section describes the process of Knowledge Representation in Neuro-Expert.

Representing knowledge in a computer consists of setting up a correspondence between a symbolic reasoning system and the outside world. This knowledge can be studied and understood in what we may call human terms, because the symbols used for its representation are seldom numerical. For example a Shares Advisor may use the following rule.

If the demand is good the company's share is good.

Such a rule would be given in the program's knowledge base in the following form.

(#rule1:

(demand = good (0.67)) → (company = good (0.87))

The knowledge representation is done through a procedural language which is called as a Knowledge Definition Language and it is constructed according to a set of rules. This set of rules constitute the Grammar of the language.

6.2.1 Knowledge Definition Language (KDL)

The Knowledge Definition Language is the knowledge representation language used in Shell. This is a user friendly Logic programming language. The structure of the language is modular in nature. There can be any number of modules as long as there is enough memory with the system.
A limited number of built-in-functions are provided as a part of the language.

The following part of this chapter describes the **KDL** in detail.

1. **Character set**: KDL uses the letters A to Z (both upper and lower case), the digits 0 to 9, and certain special symbols as building blocks to form basic program elements (numbers, identifiers, expressions etc.)

The special symbols are listed below:

\begin{verbatim}
+ : <= ]
- ; > { *
, >= }
/ " != #
:- . ( @
= - ) !
< [ & | 
\end{verbatim}

2. **Identifiers**: An identifier is a name that is given to some program element, such as variables, modules or main module. Identifiers are comprised of letters and digits, in any order, except that the first character must be a letter. Both upper and lower case are permitted and are considered to be indistinguishable. Under score can be used between any two characters as
connector. The maximum length of an identifier is 12 characters including connectors if any.

(3) **Numbers:** Numbers can be written in several different ways in KDL. In particular, a number can include a sign, a decimal point. Scientific notation is also allowed.

The following rules apply to all numbers.

(1) Commas and blank spaces cannot be included within the number.

(2) The number can be preceded by a plus sign (+) or a minus (-) sign if desired. If a sign does not appear, the number will be assumed to be positive.

(3) Numbers cannot exceed a specified maximum and minimum values. The range is 1.7E-308 to 1.7E+308.

(4) **Strings:** A String is a sequence of characters (i.e., letters, digits and special characters) enclosed by double quotes ("). Both upper and lower case can be used. The maximum number of characters that can be included in a string is 255 which is adequate for most purposes. Within a string only single quotes (') are allowed.

(5) **Data types:** One of the most important and interesting characters of KDL is its ability to support two data types. They are simple data type and compound data type.
Simple data type are single items that are associated with single identifiers on a one-to-one basis. There are three single data types. They are numeric, data, and boolean. An identifier with a data type numeric can hold a real number of any form. An identifier with a data type can hold a valid date of ten character length. The identifier with a boolean data type can hold a TRUE of FALSE value. Boolean type data are truth values that are either true or false.

Compound data type consist of multiple items that are related to one another in some specific manner. Each group of items is associated with one identifier. The KDL supports string data type. The identifier with a string data type can hold a string of 255 character length.

(6) Constants: The KDL has two built in constants, viz. TRUE and FALSE. These hold a True/false value or Yes/No value. These constants can be used in the statements to initialize a boolean type identifier. The user will not be allowed to redefine these constants.

(7) Variables: An identifier whose value is allowed to change during the execution of the knowledge is called a variable. The data type of the variable will be automatically determined by interpreter depending on the first usage of the variable. Later it will not be allowed to use as another data type.

For example, if a variable say DATE is used in a date function, then variable DATE will be considered as a
date type data. It cannot be referred to as numeric, boolean or string data type later in the knowledge base.

The data type of database fields will be kept as such through a consultation. Data type conflicts between user defined variable and database fields are not allowed.

(8) **Rule number:** The rule number is label which identifies each rule. The rule number, comprises of letter, digits and hyphen. Both upper and lower case letters are allowed. The rule number should start with character hash (#). The ending character cannot be a hyphen and two or more consecutive hyphens are not allowed. The maximum length of a rule number is 12 character including the hyphens. A rule number should be unique throughout a knowledge base.

eg: #23-03-A, #ABCDE-001

(9) **Statements and assignments:** The KDL statement can be a function, arithmetic statement or a group of rules. There are two basic types of statements in KDL, viz., simple and compound statements. The simple statements are essentially single, unconditional instructions that perform one of the following tasks.

(1) Assign a data item to a variable or assign an expression to a variable. This is called an assignment statement.

(2) Access to a system function.

(3) Access to a module.
Some typical examples for assignment statements

ASCST: - TRUE
GPAY: - BASIC + DA + HRA;

(10) Functions/commands: KDL contains a number of standard functions that are used with various data types. These functions can also be called as built-in functions. All the functions return TRUE on success and FALSE on failure. Some of these functions accept parameters.

Variable must be assigned, some value using these functions, before using them within the RULES statement.

Example: 1 REFERDB "mydata.dbf", EMP_NO; This function refers to the database file MYDATA.DBF and loads the record using the value of key field EMP_NO obtained from the user.

Example: 2 ASK "Are you a permanent employee" to PERMANENT;

The ASK function prompts the user and accepts the TRUE or FALSE value and stores in the variable PERMANENT.

(11) Rules: Basically the knowledge base is constituted of different modules. All the modules have the same status except main module. When the knowledge base is executed, the main module guides the inference engine through the other modules.

A module is constituted of different statements which can be arithmetic expressions, function calls (no user defined functions are allowed) and rules.
To every legal statement, called a premise of proposition, one of the two possible values TRUE and FALSE is assigned; these are often called Boolean Values, after the mathematician and logician George Boole (1815-1864). Complex propositions can be expressed by using logical connectives written as follows.

\[
\begin{align*}
\text{AND} & \quad \& \\
\text{OR} & \quad | \\
\text{NOT} & \quad ! \\
\end{align*}
\]

The Inference Engine expects either TRUE or a FALSE value from a statement and if the returned value is TRUE execution continues to the next statement else execution of the knowledge base is stopped.

Rules of nodes in the knowledge base are represented as follows:

\[\text{#<rule number>:} \]

i) Rule number is an alphanumeric string followed by a colon (:) . The allowed separator is hyphen (−). Rule number should not start or end with the separator.

Rule number length should not exceed 11 characters including the separator. Spaces are also not allowed inside a rule number. Rule numbers should be unique in any Knowledge Base.

ii) Conditions are expressions which evaluate to a boolean value. Each condition should be enclosed within brackets and must specify the premise strength in round brackets.

\[\text{e.g.} \quad \text{(demand = OK (0.5))} \]
The comparators comprise of the following:

- `<`  less than
- `>`  greater than
- `=`  equal to
- `<=` less than or equal to
- `>=` greater than or equal to
- `!=` not equal to

Variables can be numeric, string or boolean types depending on the function used to obtain the value. The variables should have a value before it is being used in a c.f. explanation range (0 to 1.0) etc.

The knowledge representation language is designed in such a way to avoid the complexities of the conventional languages in developing expert system applications. The structure of the Knowledge definition language is more like that of a Fourth Generation Language (4GL). This gives the users more flexibility and easy means to represent their logic.

iii) Conclusions: The conclusions are specified after the reserved "→" symbol and are specified in round brackets.

   e.g. (advice = "Buy the share")

iv) Rule Strength: The rule strength must be specified at the end of rule specification enclosed in round brackets. The rule strength would be a float value in the range -1.0 and +1.0.
The following are the built-in-functions provided in the Knowledge Definition Language. Each of the function returns a TRUE or FALSE value to the Inference Engine depending on success or failure. A failure will force the Inference Engine to stop execution. No user defined functions are allowed except that external executable programs can be executed with the built-in-functions CALL.

*ASK    *CALL    *DATEDIFF    *GETDATE
*GETNUM  *GETSTRING  *GETSYSDATE  *WRITE
*MODULE  *RUN   *REFERDB   *LOAD

A brief discussion on the above functions follows:

(1) ASK

**USAGE** : ASK "prompt" to <var name>

**PARAMETERS**

"prompt" : The literal text (enclosed in quotes) can be displayed by the ASK clause.

Variable : Any valid variable name. If the variable is a new one, then it will be considered as a BOOLEAN type.
DESCRIPTION  The ASK statement displays its prompt message to the user, then waits for a response which is in the form of YES OR NO. The value entered by the user is assigned to the variable.

(2) CALL

USAGE          CALL <"filename">

PARAMETERS

filename     The executable program name within double quotes.

DESCRIPTION  The CALL clause executes a DOS executable file (viz. files with .com, .exe, .bat extensions).

(3) DATEDIFF

USAGE        DATEDIFF <DATEDIFF <Datel>, <Date2> to <Variable>

PARAMETERS

Datel          The variable should be of type Date and it should contain a valid Date.

Date2          The variable should be of type Date and it should contain a valid Date.

Variable      The difference between the dates is assigned to the variable and it should be a numeric type.
DESCRIPTION
Finds the difference between the two given dates and stores the number of days in the given numeric variable.

(4) GETDATE

USAGE
GETDATE <"prompt"> to <variable>

PARAMETERS

"prompt"  The prompt string (enclosed in quotes) to be displayed.

Variable  If the variable is new one, then it will be considered as a Date type. Otherwise it should be DATE type.

DESCRIPTION
The GETDATE statement displays its message to the user, then accepts the date and assigns to the variable. If the user enters an invalid date then an error indicating beep sound is produced and the system prompt to enter a valid date.

(5) GETSYSDATE

USAGE
GETSYSDATE to <date>

PARAMETERS

date  The system date is assigned to the variable <date>
DESCRIPTION
The system date is to be set before using this function, if you do not have real time clock in the system. This function gets the date from the system and is stored in date.

(6) GETNUM

USAGE
GETNUM <"prompt"> to <variable>

PARAMETERS

"prompt"  The prompt string (enclosed in double quotes) to be displayed.

variable  Any valid variable name. If the variable is a new one, then it will be considered as a numeric type. Otherwise it should be a numeric type.

DESCRIPTION
The GETNUM statement displays its prompt to the user, then waits for a response to accept the number to be supplied by the user. The actual number entered by the user is assigned to the variable.

(7) GETSTRING

USAGE
GETSTRING <"prompt"> to <variable>

PARAMETERS

"prompt"  The prompt string (enclosed in double quotes) to be displayed.
variable Any valid string type. When defined for the first time, the variable will be considered to be string type.

DESCRIPTION The GETSTRING statement display the message to the user, then waits for a response to accept the string to be supplied by the user. The actual string entered by the user is assigned to the variable.

(8) WRITE

USAGE WRITE <" user prompt" >
[<variable>,<endln>]

PARAMETERS

User prompt Any valid message string.

Variable Any valid variable

Endln Newline

DESCRIPTION This inbuilt function allows the display of user messages and the prompts on the screen. This is the output statement in KDL and also has provisions for new lines if specified.

(9) MODULE

USAGE MODULE <module name>
**PARAMETERS**

**Module name**
The name of the module to be loaded or called

**DESCRIPTION**
This in-built function allows the knowledge engineer to split up the rules into modules in order to support modular and structured programming. Each module could be called using this function.

(10) **RUN**

**USAGE**
RUN

**PARAMETERS**
None

**DESCRIPTION**
This in-built function allows the user to run the currently loaded knowledge base.

(11) **LOAD**

**USAGE**
LOAD <knowledge base>

**PARAMETERS**
Knowledge Base : The name of the knowledge base to be loaded.

**DESCRIPTION**
This command loads the specified knowledge base. The system assumes that each knowledge base would have ".knb" as the extension.

(12) **REFERDB**

**USAGE**
REFERDB <"filename">, <key field>
PARAMETERS

Filename : The Database file name within double quotes

Key field : key field of the database record.

DESCRIPTION : The REFERDB clause stores the first record matching the key field after getting the key field value from the user.

6.2.3 Sample Knowledge Base

This sub-section presents a window into the shares knowledge base. The actual knowledge base comprises of more than 200 rules, however, we present here only some sample rules.

Rules to grade price

(#rulepr1:

(max_p - a (0.67)) &
(price = a (0.78))
→
(p_grade = a) (0.9))

(#rulepr2:

(max_p = b (0.8)) &
(avg_p = c (0.78)) &
(price < b (1.00)) &
(price >= c (1.00))

→
(p_grade = b) (0.9))

(#rulepr3

(avg_p = a (0.89)) &
(min_p = b (0.78)) &
(price < a (0.89)) & (price >= b (0.89))

→
(p_grade = c) (0.93))

Rules to grade transport cost

(#ruletc1:

(max_t = a (0.89))
(t_cost = a (0.9))

→
(t_grade = a) (0.94))

(#ruletc2:

(max_t = a (0.89))
(avg_t = b (0.89))
(t_cost < a (0.9)) (t_cost >= b (0.9))

→
(t_grade = b) (0.94))

(#ruletc3:

(avg_t = a (0.89))
(min_t = b (0.89))
(\text{t\_cost < a} \ (0.9))
(\text{t\_cost \geq b} \ (0.94))

\rightarrow

(\text{t\_grade = c} \ (0.94))

Rules to grade demand

(#ruled1:)

(\text{p\_grade = c} \ (0.89))
(\text{t\_grade = c} \ (0.89))
(\text{brand = good} \ (0.9))

\rightarrow

(\text{write "demand is very high"}] 
(\text{demand = vhigh} \ (0.9))

(#ruled2:)

(\text{p\_grade = b} \ (0.89))
(\text{t\_grade = c} \ (0.89))
(\text{brand = good} \ (0.9))

\rightarrow

(\text{write "demand is high"}) 
(\text{demand = high} \ (0.94))

(#ruled3:)

(\text{p\_grade = a} \ (0.89))
(\text{t\_grade = c} \ (0.89))
(\text{brand = good} \ (.9))

\rightarrow

(\text{write "demand is ok"}) 
(\text{demand ok} \ (0.98))
6.3 User Interface and Consult Facility

The User Interface in Neuro Expert is built-up of using the basic screen handling routines. It provides basic navigations with the help of pull-down menu structure, Context-sensitive help, Integrated Editor and basic DOS utilities.

The main menu of neuro expert: The Main Menu of Neuro Expert consists of six options. The options can also be selected either by selecting the required option from the ring menu or by pressing the appropriate hot keys as displayed at the
status bar at the bottom of the screen. The selected option will be highlighted and on selection the corresponding help message would be displayed in the message box.

The available options are

F1  File
F2  Edit
F3  Induce
F4  Consult
F6  Help
F10 Quit

Let us now have a detailed look at all the options.

6.3.1 File

The files menu allows to access DOS file commands without quitting Neuro Expert. This option displays a pop-up menu which consists of the following choices:

(a) Load
(b) Rename
(c) Directory
(d) Erase

These can be selected by pressing F1 key from the main menu. The choices can be selected using arrow keys and pressing return. The menu options can also be selected pressing the upper case letter which is displayed in each menu option. If an undefined key is pressed, the system produces a beep sound.
Each option of the menu is described below.

6.3.1.1 Load

Knowledge base should be loaded into the memory before consult or edit is invoked. Using the load option, the contents of the knowledge base file can be transferred to the system memory.

Extension for the knowledge base file name should be .KNB. Other file name extensions are not valid for the knowledge base file.
All cursor keys can be used to edit the path name. Left and right arrow keys can be used to move through the path name. The Home key takes the cursor to the beginning and the End key to the end of the path name. Del key erases the character in the cursor position and Backspace key erases the character in the left side cursor. Editing can be aborted by pressing Esc key and the system retains the old file loaded. After editing the filename Enter key should be pressed to load the file.

If the system finds errors in the path of file name or if the file is not found, it displays error messages. If the file is successfully loaded, the file name and the number of lines loaded are displayed in the title window at the top of the screen.

6.3.1.2 Rename

This option lets the user to rename a file. If it is only required to rename the file, enter a new name. The Rename option when used to rename a file.
6.3.1.3 Directory

The directory option displays the list of knowledge bases in current working directory as with the DOS command "dir *.knb"
6.3.1.4 Erase

Choosing this option lets the user delete a file from the disk. When this option is selected Neuro Expert prompts for the name of the file to erase. After selecting the file to delete, Neuro Expert erases the file from the disk. This command is equivalent to DOS command delete.
6.3.2 Edit
The edit option allows the user to edit the current knowledge base and the desired evidences. The edit option further displays a sub-menu with two sub-options:

- Knowledge Base
- Desired Evidences
6.3.2.1 KNB

This sub-option allows the user to edit the current knowledge base. The option automatically opens the corresponding ".knb" file using the integrated system editor, "NE". The user can modify the rules using the normal editing keys and save the changes and come back to the main menu.

6.3.2.2 Desired evidences

This sub-option allows the user to edit the desired evidence file for the current knowledge base. The option automatically opens the corresponding ".evi" file using the
integrated system editor, "NE". The user can modify the rules using the normal editing keys and save the changes and come back to the main menu.

6.3.3 Induce

This option would allow the user to derive appropriate CFs for the rules in the knowledge base and also indicate whether the knowledge base is consistent or not. Internally this option comprises of two phases:

- Mapping Rules to ANN (Phase I)
- Training the ANN (Phase II)

6.3.3.1 Phase I

During the first phase the rules of the currently loaded knowledge base are mapped to Nodes of ANN.
6.3.3.2 Phase II

During the second phase, the ANN is trained to achieve the desired evidences as specified in ".evi" file by the user. If the training succeeds then the current knowledge base is updated with the new premise strengths and evidences.

6.3.4 Consult

The Consult option should be selected after loading the knowledge base. The knowledge base can be loaded through the files menu. Consult option initiates the interactive session.
with the users, wherein the user gives his specifications and based on the user inputs and the trained knowledge base the inference engine fires the corresponding rules and arrives at a conclusion.

Depending on the user input i.e. the company selected in this case Neuro Expert would provide advice and also the line of reasoning behind it. For instance if the user selects ACC cement as the company name then the following advice would be flashed on the screen.
6.3.5 Help

The Neuro Expert help is a context sensitive help which gives advice pertaining to the topic from where help is requested. The help is also provided with a table of help indices. Using arrow keys and pressing return, a topic can be selected. HOME key and END key to be incorporated in barmenu function. PGUP key brings the selection bar to the top of the current column in the index window and PGDN key to the bottom of the current column in the index window.
The help can be invoked at any instant by pressing F5 key. If F5 key is pressed while in the help session the index for the help will be made available for selecting another topic. The arrows at the top and bottom right corners of the help screen indicate whether more information is available in the preceding or proceeding page. PGDN/PGUP or up arrow, down arrow keys can be used to scan through the topic selected.
6.3.6 Quit

The user can quit Neuro Expert by either selecting quit option from the main menu or by pressing F6. The system displays a confirmation box and confirms the selection. At any instant the user can quit from any pop up menu without selecting an option by pressing Esc key.

6.3.7 Neuro Expert Menu Command Tree

A graphical representation of the menus provided with Neuro Expert is shown in the following figure.