A Hybrid Expert System Framework: Coupling Expert System, Neural Network and Database Management System

R.M. Sonar
National Institute of Bank Management, NIBM PO, Pune–411 048 (India)

H.V.Sahasrabuddhe
Department of Computer Science, University of Pune, Pune–411 007 (India)

A.Saha
National Institute of Bank Management, NIBM PO, Pune–411 048 (India)

Abstract

In this paper, a new class of hybrid expert system framework using relational database systems as integrating mechanism has proposed, designed and implemented to develop expert systems and neural networks, and, to couple them to through the database systems using client/server model and blackboard model of problem-solving. A new coupling approach (database coupling) is used to integrate expert systems and neural networks. It provides the flexibility in coupling expert systems and neural networks in different integration modes in single-user and multi-user environment. The intelligent systems are themselves stored in relational database to be concurrently accessed and shared by multiple users. The database triggers have been used to link hypotheses and to implement the control component. The resulting framework is modular, extensible, interactive and user-friendly.

Keywords: Hybrid Intelligent System, Expert Systems, Neural Networks, Client/Server Systems, Blackboard Systems

1. Introduction

The intelligent systems have been successfully put into use to address various research and practical applications in various domains. The recent trend has been towards integrating more than one technique to solve a problem, as it may not always be possible to address all of the problems by using a single technology [9,12-13,14,23,24,29,30]. There are many applications coming up where expert systems (ESs) and artificial neural networks (ANNs) have been integrated to solve a problem [4,12,28,34,36]. The hybrid approach has been found suitable to enhance the strengths of individual technique and overcome the weaknesses, thereby addressing more complex problems. With emerging interest in hybrid systems of ANNs and ESs from both academic researchers and commercial developers, there is growing need for tools and frameworks that can integrate ESs, ANNs as well as conventional systems like spreadsheets and database management systems. There has been many applications that require both the technologies to work together to solve a task [4,34]. The integration not only combines the strengths but also improves efficiency and accuracy of individual technique. However, their strengths and weaknesses tend to be complementary, and, for some specific applications, it is more appropriate to link them in a simple manner so that the entire system will maintain the individual strengths of two technologies.

The decisions such as financial decisions often deal with practical and real-life problems. Such decisions are quite complex, risky, require proper examination of several qualitative as well as quantitative factors, and, often require processing of large volume of data need before coming to any conclusion. Data to be processed needs to be organized properly to make it comprehensible and relevant to the task in hand. The process of analysing financial data consists of several steps. In general, the analyst has to decide the objectives and perspective of the analysis (e.g. as creditor, investor etc.). For financial ESs, it is likely that a large amount of data will be repeatedly analyzed. This makes a sequential question-and-answer process very slow, tedious, expensive and impractical. Lot of recursive computations such as financial ratios and their combinations are involved before consideration of the information as input for analysis. That makes the systems data intensive, requiring extensive input/output (I/O) operations and exchange of information. A better approach in such cases is to design the intelligent systems having efficient database interface to connect to the database systems. Database systems effectively and efficiently manage large database of information. An SQL (Structured Query Language) interface can help to select, organize and transform the data dynamically. SQL can be used to retrieve required information for processing depending on the task in hand. Data intensive nature of financial decision making
Various strategies have been suggested to integrate ES and ANN [3,8,12,14-18,35]. Depending upon the medium of exchange of information between ES and ANN, the hybrid systems can be broadly classified as loosely coupled, tightly coupled, and, fully integrated. The loosely coupled systems are separate and communicate through data files while tightly coupled systems communicate through internal memory buffers [14]. Fully integrated systems share the internal data structures and presentations [14]. Many of the hybrid applications use ESs and ANNs as separate components [4,6,34]. These applications use loose coupling and tight coupling approaches to integrate the systems. This is because it is more appropriate to integrate ESs and ANNs in a simple manner so that entire system maintains the individual strengths and they can easily be developed using commercially available tools. However, these approaches may not be appropriate when the applications are data intensive requiring extensive information exchange, and, requiring flexibility in organizing, selecting, and, manipulating the information while exchanging/applying the information between/to the integrated systems. It may be difficult to develop loosely and tightly coupled hybrid systems in multi-user environment rather than in a single-user environment as the issues like concurrency, integrity and consistency of information need to be taken care of while developing the hybrid systems.

The client/server computing model delivers the benefits of network computing model along with shared data access and high performance characteristics of the host-based computing [1,7]. Database servers in client/server computing model can manage a single database of information among many concurrent users, control the database access and other security requirements. In such an environment, intelligent systems (henceforth, the term intelligent systems is referred to ESs and ANNs) themselves, as well as the data they access can be stored in relational database to be accessed by multiple clients. This approach is different than the way the most of the commercial expert system shells and neural network development tools store intelligent systems at the client-side.

The blackboard model of problem solving [2,5,20,22] is suitable when a great deal of flexibility in knowledge and problem-solving representations along with the dynamic control of problem-solving activities is required. The client/server computing and blackboard model of problem solving have some basic similarities: 1> the relational database resembles the blackboard, 2> client applications as knowledge sources, and, 3> the database server as the blackboard server. A new framework can be thought that can take advantage of client/server computing and blackboard model of problem solving to integrate intelligent systems to build hybrid applications that are data intensive and require distributed processing. This framework would make the intelligent systems to share the database as well as to communicate through the database.

In this paper, we present a hybrid expert system framework Relational Blackboard Framework (RBF) that tightly couples intelligent systems to the relational database server and uses the same relational database server as an integrating mechanism to couple and control intelligent systems. This framework can also be used to develop intelligent systems and integrate them with other applications. The term hybrid expert system is referred to the integration of ESs with systems like ANNs and database systems [8,36]. We also discuss how RBF can support different modes of integration like sequential (chainprocessing), parallel, embedded(subprocessing), metaprocessing and coprocessing [27,18] in a single-user as well as in multi-user environment.

2. RBF Concept

The Relational Blackboard Framework (RBF) proposed in this paper incorporates the functionality of blackboard model into client/server architecture. Figure 1 shows the general concept behind the RBF. The RBF uses a relational database as a blackboard (relational blackboard) and C/S computing model as a blackboard model (relational blackboard model). The relational database server is used as blackboard server (relational blackboard server) that manages the relational blackboard.
The intelligent systems and other applications are independent knowledge sources share the data and communicate through the relational blackboard as shown in figure 2. The intelligent systems are stored in relational blackboard. A requested ES or ANN is fetched from the blackboard and executed. This eliminates the need to store ESs and ANNs on client machine/s. Storing the ESs and ANNs in relational blackboard helps to share and have concurrent access to the data as well as intelligent systems. The information in conventional blackboard is stored in levels or regions and data in one level may be linked to next level. In relational blackboard, data is stored in tables and can be associated using views and linked through database triggers. View is virtual table in which data from base tables are combined [1].

The database recordsets and their types play a very important role in the exchange of information in C/S environment. A recordset (RS) is a set of records retrieved from the database in response to an SQL-query coupled with a cursor. A cursor enables to navigate through the set of records. The information among the intelligent systems and other applications is exchanged using the recordsets. Figure 2 shows the linked recordsets to the intelligent systems and how they share the data through linked recordsets. The recordsets are linked externally eliminating the need to write the code for database operations inside the ES or for ANN I/O interface. A recordset can be refreshed with new contents by sending a new query. The mapping between recordset columns and intelligent system variables enables the data input and output between the intelligent systems and relational blackboard.
Instead of opening a recordset and fetching a row at a time, client-side batch recordsets are used to reduce network traffic and increase overall performance. Batch recordset and batch updating help to improve performance by locally caching changes to data and then writing them all to the database server in a single update. In RBF, the set of rows (batch) is a major unit of exchange of the information among the KSs. When KSs need to share the data they can open recordsets having common database portion. The KSs can also share the parameters through relational blackboard.

Coupling intelligent systems through linking recordsets shows a type of coupling, which is different from tight and loose coupling. This can be termed as database coupling. The definition of database coupling can come in between loose coupling and tight coupling. When a client-side batch recordset is open, it acts as a memory buffer (can be treated as a form of tight coupling) and when changes are sent in a batch, they are reflected in another linked recordset through the database server (can be treated as a form of loose coupling). When ES and ANN both are linked to the same recordset, they can be said as a tightly coupled. The dotted line in this figure 3 shows the indirect database coupling between ES and ANN. Multiple ESs and ANNs can be coupled using database coupling.

The control in RBF is event based like in HASP/SIAP blackboard architecture [2] and database triggers are used to implement it. The scheduling logic is implemented using database triggers. In most of the blackboard systems, the blackboard changes are treated as events [2]. In relational blackboard, these changes are execution of data manipulation operations such as insert, delete or update. A database trigger can be used to perform an action whenever some data manipulation operation takes place.

In C/S architecture, the KSs are GUI applications. A KS may respond to many GUI events like mouse click, menu-selection etc. and perform corresponding functions. An instance of a KS can be in running mode waiting for an event to happen unlike in conventional blackboard systems where a knowledge source instance (KSI) is executed only once i.e. load, execute and exit, when it is scheduled for execution. Therefore the control mechanism required in such a framework is different from control in conventional blackboard systems. The events are mapped not only to KSs, but also to the events to which they respond while scheduling the activities. Sunners et al [27] have described an interactive framework that uses GUI based knowledge sources. They have used object-oriented approach to implement the blackboard framework. In RBF, there are two types of events to which a KS can respond: user-event and control-event. User-events are GUI events that occur when the user interacts with the KS. Control-events are generated by the control component to execute a KS or some function/s inside the KS.

The relational blackboard stores the pending KSI s and events (KSI Queue) in a table (control table) as shown in figure 1. This table is shared among all running KSs. When a KSI is idle, it itself looks into the control table for any pending events to be executed. An event may terminate the KSI itself. Whenever a hypothesis is posted, the

Fig. 3 The database coupling approach to integrate ES and ANN

4 Hybrid Expert System Framework, January 2000
database trigger activates and makes corresponding KS or event within that KS as pending KSI to execute next. The scheduling of KSs can be done in such a way that no more than one KSI can simultaneously update the same data, and, no redundant execution of KSs.

3. RBF Implementation

The RBF framework described in section 2 has been implemented with Windows 95 [33] as client platform and Oracle [21] as a relational database server. Any relational database server other than Oracle can be used. Three types of knowledge sources are considered in this framework: 1> intelligent KS (ES and ANN), 2> executable KS, and, 3> spreadsheet KS. Hybrid Expert System Shell (HESS) is a major component of the overall framework that facilitates the development as well as integration of ES and ANN. An executable knowledge source (EKS) is a Windows executable file and a spreadsheet knowledge source (SKS) is a spreadsheet file. While HESS manages the execution and integration of intelligent knowledge sources (IKS) through the blackboard, an EKS itself manages interaction with relational blackboard. The SKS execution and integration is managed through spreadsheet package (in this framework MS-Excel [19]).

3.1 Hybrid Expert System Shell

The HESS is implemented using Visual C++ [32] is the extended version of hybrid shell described by Sonar [25]. It is basically a client application. Many instances of the HESS can be running on different machines or same machine. It comprises five components as shown in figure 4. These components are:

1. Expert System Shell (ESS),
2. Neural Network Module (NNM),
3. Database Interface (DBI),
4. Command Language Interface (CLI), and,
5. Control Module (CM)

The ESS and NNM are separate and independent. ESS is a rule-based expert system shell to develop ESs. NNM is an ANN module to develop ANNs using backpropagation architecture. The DBI facilitates seamless integration of ESS and NNM to the relational blackboard. The DBI provides the list of ESs and ANNs available in the relational blackboard. A requested ES or ANN is fetched from the blackboard and executed. The DBI manages opening and linking of the various recordsets to the intelligent systems externally.

![Fig. 4 Inside hybrid expert system shell](image)

The HESS can be used in two modes: interactive and auto mode. In interactive mode, the functioning of the HESS is controlled by the user such as loading ES or ANN, connecting to the database server and opening the database, opening and linking various recordsets, executing stored procedures, executing SQL statements etc.
In auto mode, there is no user interaction with HESS, the CLI and CM components control HESS operations. The CLI is a procedural language interface to execute the commands that are executed by the user in interactive mode. The CM controls and coordinates the activities in multi-user environment.

**Expert System Shell**

The ESS has all the features that a typical rule-based ES shell has with backward and forward reasoning. User can select the ES name and DBI brings entire ES into the local rule editor from the relational blackboard. Subsequently, the user can load it into memory for execution. Figure 5 shows the names of the database tables that store the ESs. The ES can be run in interactive question-answer mode or it can directly take input from or store output to the relational blackboard through the linked recordset. A recordset can be linked (input recordset) to the ES by firing an SQL-query or selecting table/view name with the help of DBI. When no recordset is linked, the ESS automatically runs the ES in question-answer mode. When a recordset is linked, ES runs in batch-mode i.e. it runs for the number of sessions equal to the number of records in the recordset, starting with the first record. A record pointer is used to keep the track of records in recordsets. It is initially set to the first record before start of the batch-run and incremented after each session until the last record.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert System Table</td>
<td></td>
</tr>
<tr>
<td>ES_ID</td>
<td>Unique identification number for ES</td>
</tr>
<tr>
<td>ES_NAME</td>
<td>Name of ES</td>
</tr>
<tr>
<td>GOALS</td>
<td>Goals of ES</td>
</tr>
<tr>
<td>VARIABLES</td>
<td>Variables used in the ES</td>
</tr>
<tr>
<td>INPUTS</td>
<td>Input variables to be linked to the RS</td>
</tr>
<tr>
<td>OUTPUTS</td>
<td>Output variables to be linked to the RS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule Table</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ES_ID</td>
<td>An unique identification number for ES</td>
</tr>
<tr>
<td>RULE_ID</td>
<td>An unique identification number for rule</td>
</tr>
<tr>
<td>CONDITION</td>
<td>The condition part of rule</td>
</tr>
<tr>
<td>ACTION</td>
<td>The action part of rule</td>
</tr>
<tr>
<td>ELSE_ACTION</td>
<td>The ELSE action part of the rule</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>The description of the rule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression Table</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ES_ID</td>
<td>An unique identification number for an ES</td>
</tr>
<tr>
<td>LINE_ID</td>
<td>line number of expression</td>
</tr>
<tr>
<td>EXPR</td>
<td>Expression</td>
</tr>
</tbody>
</table>

Fig. 5 Column names of database tables that store Expert Systems in RB

A column can be used for input as well as output. Two different recordsets can be linked to ES as shown in figure 6(b). One is for input variables while other one (output recordset) is for output variables. Two recordsets are needed when linked input recordset is non-updatable. Non-updatable recordset does not allow any update or delete operation on recordset [15]. Recordset can be opened and linked before ES is loaded into the memory because at run-time only the I/O between ES variables and current records takes place. Since the recordsets used are client-side batch recordsets, they are refreshed whenever the expert system starts batch-run in order to reflect the changes to the relational blackboard made by other users.

The table ES stores the entries associated with each ES. It is just like header portion ES that includes the name of ES, what goals it has to achieve and what are the variables including input and output variables used in the ES. The input variables are the variables that take input (facts) from the currently available record of linked recordset. Similarly, the output variables' (goals, subgoals, etc.) values are updated to appropriate columns of currently available record. The appearance of variable names should correspond to names of columns. The variable names and column names may be different but their order is important. For example, consider the I1,I2 and I3 are input variables and O1,O2 are output variables and the linked recordset say 'Select C1,C2,C3,C4,C5
from …’ as shown in figure 6(a). During each session-run, the ES would initialize the variables I1, I2 and I3 with the values of C1, C2 and C3 of the current record, whenever needed, and, update C4 and C5 of the current record with the values of variables O1 and O2 when the session gets over. Figure 7 shows a displayed input recordset.

The explanation of ES session can be stored into the recordset itself. If the column named as inference appears in the recordset, the ESS automatically puts the explanation of ES session into the current record. However, this column must have appropriate data type and size to store the whole explanation.

The data types of variables are determined at run-time, a variable can be used to store different types of data. In case of input variables, the data types of input variables may not match the corresponding data types of columns.
Depending upon the values of columns, the data types of input variables are determined. This makes the system independent of database schema. In case of output variables, the values of output variables should match with the data types of columns because the values are updated to the recordsets.

RULE table stores the rules for all ESs. Each rule has an ES_ID, which stores ID of ES to which the rule is associated with, and, a RULE_ID to identify the rule itself. The syntax of rule is IF logical expression THEN expression ELSE expression. EXPRESSION table stores the expressions or the statements other than rules used in ESs. The statements are executed before shell starts execution of rules. These expressions may include menu templates, questions, initialization statements, assignment expressions etc.

Other distinguishing feature of ESS is that it provides various database functions to dynamically execute the stored procedures, call database built-in functions, and, execute DDL (Data Definition Language) as well as DML (Data Manipulation Language) SQL statements. The database functions can be included anywhere in the ES including in expressions, condition part or action parts. Four major database functions that are allowed inside ES are described in table 1. Most of the RDBMS provide rich set of built-in functions for date manipulation, data conversions, data formatting, mathematical calculations, user environment information etc. to perform various tasks. Calling these functions directly inside the ES eliminates the need to implement such type of functions in the shell itself. These functions can also be used to keep the track of security aspects such as user identifications, database session numbers, timings etc.

Table 1: The database functions supported by ESS

<table>
<thead>
<tr>
<th>Function Name</th>
<th>What it Does</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL(SQL-Statement)</td>
<td>Sends the SQL-statement (excluding select statements) to server and returns the status, whether executed successfully or not.</td>
</tr>
<tr>
<td>STOPROC(StoredProcCall)</td>
<td>Executes the stored procedure at server side. It may return the execution status or the result.</td>
</tr>
<tr>
<td>DBFUN(FunctionCall)</td>
<td>Executes the database built-in function and returns the result.</td>
</tr>
<tr>
<td>SELECTSQL(SelectQuery)</td>
<td>Sends the select query to the server and returns a row or a set of rows.</td>
</tr>
</tbody>
</table>

The ESS supports matrix data types and their manipulations. This data type is added to embed and control ANNs inside the ES as ANN algorithms are implemented using matrix as a major object. Matrix variables can also be used as arrays (one or two dimensional). Table 2 shows a partial list of functions that can be used to control and supervise an ANN through ES. Inside ES some predefined global variables to set or use the ANN parameters can be used. For example, Hidden1 is a variable is used to set/get the number of neurons in hidden layer 1. Table 3 shows the list of predefined global variables. After setting the parameters, the ES can use Initialise() function to initialises the ANN topology by taking the current parameters.

Table 2: Functions to control and supervise ANNs:

<table>
<thead>
<tr>
<th>FunctionName(Parameter/s)</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadAnn(AnnName)</td>
<td>Fetches ANN from RB and loads it into memory</td>
</tr>
<tr>
<td>RunAnn(AnnName,InputMatrix)</td>
<td>Runs the ANN by taking input from specified matrix</td>
</tr>
<tr>
<td>Train(Iterations)</td>
<td>Trains the ANN for number of iterations</td>
</tr>
<tr>
<td>TrainMomentum(Iterations)</td>
<td>Trains the ANN for number of iterations using momentum and learning rate adaptation algorithm</td>
</tr>
<tr>
<td>TrainSA(Iterations)</td>
<td>Trains the ANN for number of iterations using simulated annealing algorithm</td>
</tr>
<tr>
<td>Initialise()</td>
<td>Initialises the network with current parameters</td>
</tr>
<tr>
<td>SaveAnn()</td>
<td>Stores the network parameters and connection weights</td>
</tr>
</tbody>
</table>
### Table 3: ES global variables and ANN parameters

<table>
<thead>
<tr>
<th>VariableName</th>
<th>ANN Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRate</td>
<td>Learning rate</td>
</tr>
<tr>
<td>Hidden1</td>
<td>Number of neurons in hidden layer 1</td>
</tr>
<tr>
<td>Hidden2</td>
<td>Number of neurons in hidden layer 2</td>
</tr>
<tr>
<td>ASSE</td>
<td>Average sum of the squared error</td>
</tr>
<tr>
<td>Layers</td>
<td>Number of layers in the ANN</td>
</tr>
<tr>
<td>TrFunctions</td>
<td>Transfer functions</td>
</tr>
</tbody>
</table>

### Neural Network Module

This is an implementation of back-propagation architecture. The neuron model is represented by matrix object. All the algorithms are implemented using matrix as a major object. The matrix library is developed covering almost all the major matrix functions and mathematical operations using C++ operator overloading [26]. The library also includes the functions to tightly couple matrix variables to the recordsets. Use of matrices makes it easier to implement ANN algorithms and couple ANN with the recordsets/part of recordset. Figure 8 shows the matrix operations with the recordset.

![Matrix operations on recordset](image)

**Fig. 8 Matrix operations on recordset**

Input vectors, desired output vectors, output vectors and connection weights are stored in matrices. The ANNs learn in batch mode. Therefore, the entire set of training instances is stored into matrices. **Input matrix (I), desired output matrix (T) and output matrix (O)** store the input vectors, desired output vectors and output vectors respectively. Only one recordset (training input recordset) can be linked to ANN, from which matrix I and T would read and to which O would be written. For example, consider an ANN to train XOR (exclusive OR) function. Training examples are stored in a table say XOR (inputs X,Y and desired output Z). This table may need one more column (which is optional) to store calculated output say OUT. Figure 9(a) shows how the matrix I and T, read the corresponding columns from recordset when ANN is initialized. After completion of the specified number of training iterations, the matrix O would be written into column OUT of XOR table. Figure 10 shows a linked training input recordset.

![Mapping recordset columns with ANN Variables](image)

**Fig. 9 Mapping recordset columns with ANN Variables**

9 Hybrid Expert System Framework, January 2000
Depending upon the number of input/output neurons and number of records in linked recordset, the matrix $I,T$ and $O$ are automatically resized and read/write the corresponding columns from/to linked recordset. In XOR example, number of input neurons ($r$) are 2, number of output neurons ($s$) is 1. The number of rows in $I,T$ and $O$ are set equal to number of rows ($q$) in linked recordset. Therefore, the matrix $I$ is be resized to $<r,q>$, matrix $T$ to $<s,q>$ and matrix $O$ to $<r,s>$. When network is initialised, the matrix $I$ would read first two columns ($X$ and $Y$), $T$ would read the next one ($Z$), and, after the specified number of iterations, $O$ would be written to next one column ($OUT$). The total columns in the linked recordset should be equal to $r+s+s$. If the linked recordset contains only $r+s$ columns, the matrix $O$ is not written into the linked recordset and the linked recordset is automatically opened in read only mode.

Instead of a single recordset, two different recordsets can be linked to ANN. This is needed when input recordset is non-updatable. A separate recordset (training output recordset) can be used to write matrix $O$. The number of rows in both the recordsets should match. Figure 9(b) shows how two recordsets are linked.

The NNM facilitates online testing, where test set can also be evaluated while training is going on. In online testing mode, three more matrices are used: test input matrix ($I_T$) to store test input vectors, test desired output matrix ($T_T$) to store test desired output vectors and test output matrix ($O_T$) to store test output vectors. One or two recordsets can be linked to ANN for test matrices ($I_T$, $T_T$ and $O_T$) similar to training matrices ($I,T$ and $O$) discussed above. Overall, four different recordsets can be linked to an ANN at training and testing.

Fig. 10 An ANN with linked training input recordset

ANN parameters and weights are stored in relational blackboard. The DBI provides list of ANNs available. Upon selection, the DBI fetches and loads ANN into memory. Table 4 shows ANN parameters that are stored in table ANN. The transfer functions are stored using first character (S-Sigmoid, T-ArcTangent, L-Linear etc.). For example, if the hidden layer uses sigmoid function and the output layer uses linear, then the column TRFUNCTION would contain the string SL. ASSE is Average Sum of the Squared Error calculated by dividing SSE by number of training instances. The parameters of ANN can be set interactively and saved into ANN table (table 4).
Once the network is trained, its weights are stored in the table specified in WEIGHT_TABLE column. Weight table is single column table containing connection weights in a sequence. Parameters LEARNING_RATE, ITERATIONS are not used at run-time, they are used while training. All other parameters are used while training as well as at run-time.

Table 4: ANN table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN_ID</td>
<td>An unique code for ANN</td>
</tr>
<tr>
<td>ANN_NAME</td>
<td>Name of ANN</td>
</tr>
<tr>
<td>LAYERS</td>
<td>Number of layers</td>
</tr>
<tr>
<td>TRANSFUNCTIONS</td>
<td>Transfer functions</td>
</tr>
<tr>
<td>INPUTS</td>
<td>Number of input (neurons)</td>
</tr>
<tr>
<td>OUTPUTS</td>
<td>Number of outputs (neurons)</td>
</tr>
<tr>
<td>H1_NEURONS</td>
<td>Number of neurons in first hidden layer</td>
</tr>
<tr>
<td>H2_NEURONS</td>
<td>Number of neurons in second hidden layer</td>
</tr>
<tr>
<td>WEIGHT_TABLE</td>
<td>Name of the table where connection weights are stored</td>
</tr>
<tr>
<td>LEARNING_RATE</td>
<td>Learning rate</td>
</tr>
<tr>
<td>ITERATIONS</td>
<td>Number of training iterations</td>
</tr>
<tr>
<td>ASSE</td>
<td>Average Sum of the Squared Error</td>
</tr>
</tbody>
</table>

Command Language Interface

The command language is a small set of instructions and programming constructs. The CLI facilitates the execution of various HESS operations using command language. It is a run time environment to integrate the ES and ANN inside HESS. Command language instructions include functions to (1) open and link various recordsets to ES and ANN (2) fetch ES/ANN from relational blackboard and load them into memory (3) execute ES or trained ANN (4) post the hypotheses (5) train ANN (6) handle database transactions. Table 5 shows the partial list of functions supported by the CLI. The syntax of command language instructions is simple and easy to understand. Instructions are executed in interpreter mode. The set of instructions is called as script. The use of script allows multiple instances of the same ES or ANN to be executed simultaneously in multi-user environment. However, only one instance of ANN is allowed when it is being trained. The scripts can also be stored in tables SCRIPT and STATEMENT and are brought into script editor by invoking the DBI, and, subsequently can be executed. The column names of tables are shown in figure 11.

Table 5: List of some of the functions used in scripts

<table>
<thead>
<tr>
<th>FunctionName(Parameter/s)</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpertInput(InputRS)</td>
<td>Opens and links input RS to ES</td>
</tr>
<tr>
<td>ExpertOutput(OutputRS)</td>
<td>Opens and links output RS to ES</td>
</tr>
<tr>
<td>RunExpert(ESName)</td>
<td>Fetches ES from RB and runs (backward chaining) it in batch-mode</td>
</tr>
<tr>
<td>AnnInput(InputRS)</td>
<td>Opens and links training input RS to ANN</td>
</tr>
<tr>
<td>AnnOutput(OutputRS)</td>
<td>Opens and links training output RS to ANN</td>
</tr>
<tr>
<td>AnnTestInput(InputRS)</td>
<td>Opens and links testing input RS to ANN</td>
</tr>
<tr>
<td>AnnTestOutput(OutputRS)</td>
<td>Fetches trained (run-time) ANN from RB and runs it</td>
</tr>
<tr>
<td>RunAnn(AnnName)</td>
<td>Posts the hypothesis</td>
</tr>
<tr>
<td>PostHypothesis(Hypothesis)</td>
<td>Posts the hypothesis whenever the specified number of training iterations are completed.</td>
</tr>
<tr>
<td>LoadAnn(AnnName)</td>
<td>Fetches ANN from RB and loads it into memory for training</td>
</tr>
<tr>
<td>Train(Iterations)</td>
<td>Trains the loaded ANN for next specified iterations</td>
</tr>
<tr>
<td>TrainMomentum(Iterations)</td>
<td>Trains the loaded ANN for number of iterations using momentum and adaptive learning rate algorithms.</td>
</tr>
<tr>
<td>TrainSA(Iterations)</td>
<td>Trains the loaded ANN for number of iterations using simulated annealing algorithm</td>
</tr>
<tr>
<td>Begin()</td>
<td>Begins the database transactions</td>
</tr>
<tr>
<td>Commit()</td>
<td>Commits the database transactions</td>
</tr>
<tr>
<td>RollBack()</td>
<td>Rollbacks the database transactions</td>
</tr>
</tbody>
</table>
Control Module

A control module is controlling mechanism to control the activities in client/server environment. The control module constantly looks for the event to execute the scripts, train the network or run the ES.

Database Interface (DBI)

The database interface (DBI) is a major component that facilitates seamless integration of intelligent systems with the relational blackboard. The DBI is built to fully utilize the capabilities of relational database servers. Its job is: (1) to connect to the database, (2) to provide the list of available Scripts, ESs and ANNs in the database, (3) to retrieve requested ES or ANN from database, (4) to execute stored procedures, (5) to call database built-in functions, (6) to handle database transactions, and, (7) to select, open and link various types of recordsets to ANN or ES for input or output or both.

The DBI is highly interactive and provides information about database schema such as table/view names, stored procedures and their parameters, parameter types etc. The contents of various open recordsets can be displayed and are automatically refreshed. The DBI is based on Microsoft's ActiveX Data Objects (ADO) which support key features for building client/server and Web-based applications [15]. The DBI works by accepting commands from ES/CLI or interactively using menus and dialog boxes. Once the connection to the database is successfully established, the DBI brings and stores all information about database schema into local memory. This includes table/view names, stored procedures available, what parameters stored procedures use, their types, etc. This database schema information can be refreshed, if new tables or procedures etc., are added to the database.

An executable knowledge source (EKS) is Windows executable file, having the database interface and control module integrated into it. Any instance of HESS can start the execution of an EKS when it is ready for execution. Once EKS instance starts, it itself manages the interaction with relational blackboard.

![Fig. 12 Inside executable KS](image-url)
Figure 12 shows the inside view of a typical executable KS used in the framework. A function F may be executed in response to a user event or control event and may post the hypothesis. When user interacts with HESS, the EKS may execute some function in response to the user event. When the KSI is idle, the control component looks for any pending control event/s, and, if any event is pending, the EKS executes the appropriate function. A control module can be implemented using a timer control. When KSI is idle, a timer can be enabled to look for the pending events otherwise it can be disabled.

3.3 Spreadsheet Knowledge Source

The database interface and control module can be added to the MS-Excel workbook file similar to that of EKS using Visual Basic for Applications (VBA) language. The database interface can bring the data from relational blackboard into the specified worksheet and can update the data from worksheet into relational blackboard.

3.4 Inside Relational Blackboard

Figure 13 shows the inside view of relational blackboard. The ES, RULES, EXPRESSION, SCRIPT, STATEMENT and ANN tables store ESs, ANNs and scripts. The hypothesis table contains list of posted hypotheses. It consists of a single column Hypothesis. The database trigger hypothesis trigger is executed whenever hypothesis is inserted into hypothesis table. The function of hypothesis trigger is to map the hypotheses to the events within KSs. This mapping information is stored in event table. Table 7 shows columns of event table, and, table 8 shows the event table with example entries. Hypothesis Continue Sine Training is mapped to KS SINE and event TRAIN_NEXT. The application specific scheduling knowledge can be embedded into the trigger. Other database triggers can be written to map the database changes to the execution of knowledge sources. Hypothesis trigger makes changes to the control table instead of triggering the KSs. Table 9 shows the columns of control table, and, table 10 shows the example entries.

The HESS, EKSs, and, SKSs themselves manage interaction with control table through control module. The control of activities depends on how knowledge sources and events are scheduled. Once the knowledge sources/events are scheduled for execution by putting them into control table, they are executed on first-come-first basis. However, a column priority can be added to the event and control table, so that the priority can be specified while executing a particular knowledge source/event over others. In this case, the control table can be
indexed on priority column so that the knowledge sources/events can be executed on priority. The advantage of using triggers is that various constraints can be imposed on while executing particular knowledge source such as executing a particular knowledge source on a particular client machine. Table 11 shows the hypothesis table containing posted hypotheses.

Table 7: Event table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis</td>
<td>The hypothesis</td>
</tr>
<tr>
<td>KS_NAME</td>
<td>Name of the knowledge source to which the hypothesis is mapped</td>
</tr>
<tr>
<td>KS_TYPE</td>
<td>Type of KS (S-Script, E-Executable, X- Excel spreadsheet)</td>
</tr>
<tr>
<td>Event</td>
<td>Event to which KS responds</td>
</tr>
</tbody>
</table>

Table 8: Event table with example entries

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>KS_NAME</th>
<th>KS_TYPE</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sine Training Completed</td>
<td>GRAPH</td>
<td>E</td>
<td>REFRESH_GRAPH</td>
</tr>
<tr>
<td>Continue Sine Training</td>
<td>SINE</td>
<td>S</td>
<td>TRAIN_NEXT</td>
</tr>
<tr>
<td>Terminate Sine Training</td>
<td>SINE</td>
<td>S</td>
<td>EXIT</td>
</tr>
</tbody>
</table>

Table 9: Control table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KS_NAME</td>
<td>Name of knowledge source</td>
</tr>
<tr>
<td>KS_TYPE</td>
<td>Type of KS (S-script E-executable files and X- Excel spreadsheet)</td>
</tr>
<tr>
<td>TERMINAL_ID</td>
<td>The machine identification (ID) number on which KS should execute</td>
</tr>
<tr>
<td>Event</td>
<td>Event to which KS responds</td>
</tr>
</tbody>
</table>

Table 10: Control table with example entries

<table>
<thead>
<tr>
<th>KS_NAME</th>
<th>KS_TYPE</th>
<th>TERMINAL_ID</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINE</td>
<td>S</td>
<td>NIBM_1</td>
<td>TRAIN_NEXT</td>
</tr>
<tr>
<td>GRAPH</td>
<td>E</td>
<td>NIBM_2</td>
<td>REFRESH_GRAPH</td>
</tr>
</tbody>
</table>

Table 11: Hypothesis table with example entries

<table>
<thead>
<tr>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sine Training Completed</td>
</tr>
<tr>
<td>Terminate Sine Training</td>
</tr>
</tbody>
</table>

Linked_hypothesis table is used to store the links between various hypotheses. The hypotheses that are linked together are termed here as linked hypotheses. The links among the hypotheses are basically part-of hierarchy as shown in figure 14 (where Z, Y, X1, X2, ... are hypotheses). Table 12 shows the corresponding entries to figure 14. Hypothesis X and Y are part of hypothesis Z. The column Children of linked_hypothesis table stores how many sub-ordinate hypotheses or parts a hypothesis has. The column Parent_ID stores the identification number of parent hypothesis. When all sub-ordinate hypothesis are posted, the parent hypothesis is posted automatically by the trigger called as linked trigger. For example, when hypothesis X and Y are posted, the linked trigger would automatically post hypothesis Z into hypothesis table. Figure 15 describes the operations on linked hypotheses.

A Linked hypothesis, say \( H \) is posted and, say \( P \) is the parent hypothesis of \( H \)

The children count of \( P \) is reduced by one, and, Parent_Id of \( H \) is set to 0.

If the children count of \( P \) becomes 0 (means all the sub-ordinate hypotheses are posted), then \( P \) is posted.

Fig. 14 Part-of Hierarchy

Fig. 15 Operations on table linked table
Since, the *Children* and *Parent_ID* columns of linked_hypothesis table are changed during the execution, two additional columns are used to store their original values. These columns can be used to restore the *Children* and *Parent_ID* values to set to original part-of-hierarchy. The entries in event table and linked_hypothesis table are application specific.

<table>
<thead>
<tr>
<th>ID</th>
<th>Hypothesis</th>
<th>Children</th>
<th>Parent_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Z</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>X1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>X2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>Y1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>Y2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>Y3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

4. RBF Operations

The operations of the framework can be described in two modes: single-user, multi-user. In single-user mode, only one instance of HESS is sufficient to develop hybrid applications. In multi-user mode, many instances of HESS can be running on different machines or on the same machine.

4.1 Single User Operations

In single-user mode, the HESS can be used as a single client application to connect ANNs and ESs. The systems can be integrated by selecting the appropriate recordsets that have a common database portion among them. The columns selected may be output for some system and may be input for the other. When the first system is executed, it makes available the output for the second system. Figure 16(a) illustrates how systems can be integrated. The system A takes input from columns *C1*, *C2* and *C3* of table *test* and produces the output into the columns *C4* and *C5*. The system B takes input from columns *C4* and *C5* and produces the output into *C6*. When the system A is executed it makes available output in column *C4* and *C5*, these columns are input to system B. Figure 16 (b) shows input and output columns of integrated system.
ESs and ANNs can be integrated at run-time using various command language functions. The scripts can be written that contain commands to open and link recordsets to ESs or ANNs and execute ESs and ANNs. The number of ESs and ANNs can be executed by including them in a script. Figure 17 shows how systems can be integrated in different integration modes.

Following examples can be considered as the templates that illustrate how ESs and ANNs can be integrated in different integration mode using the scripts. In all of these examples, the query of the form ‘Select C1,C2,...from ... ’ represents an arbitrary recordset where C1,C2,... represent the column names of the recordset.

![Diagram of integration modes](image)

**Fig.17 Integration Modes supported by HESS (in a single user- mode)**

**Example 1**
This example illustrates how the systems can be integrated sequentially (chainprocessing). As shown in the figure 18 (a), a script can be written to integrate the systems sequentially. The ES X takes input from column C1 and C2 and produces output in column C4. Column C3 and C4 act as intermediate columns to store the intermediate results as shown in figure 18 (b).

```
Begin
  ExpertInput ( 'Select C1,C2 from …' )
  ExpertOutput( 'Select C3,C4 from…'  )
  RunExpert('X' )
  AnnInput( 'Select C3,C4 from …' )
  AnnOutput( 'Select C5 from …' )
  RunAnn( 'Y' )
End
```

![Diagram of connecting systems sequentially](image)

**Figure 18: Connecting Systems Sequentially**
In this fashion more than two systems can be connected sequentially. The integrated system takes input from
$C1$ and $C2$ columns and produces output in column $C5$.

**Example 2**

This example illustrates how ANNs can be embedded inside ES (subprocess). Following ES code shows how
`RunAnn` function can be used inside ES to execute ANN. `PE-CLASSIFY` is the name of trained ANN. The second
parameter of `RunAnn` is of matrix type. The matrix data types are stored in square brackets. The character `|`
is used as row separator and comma as column separator. The function `RunAnn` returns the output
matrix.

```plaintext
IF RunAnn('PE-CLASSIFY',[1.2,0.34,0.15,2]) is 1
Then Company := Good
```

**Example 3**

This example shows how the training of ANN can be controlled and supervised through ES (metaprocessing).
ES can use various global variables that can set or get the ANN parameters. Figure 19 illustrates this.

**Example 4:**

This example shows how ES and ANN can solve the same task by taking the same inputs and how their results
can be combined, by another ES. As shown in figure 20(b), ES $X$ and ANN $Y$ solve the same task by taking
common inputs (from column $C1$ and $C2$), their results are compared by ES $Z$ to produce output in column $C5$.
Figure 20(a) show the script.

```plaintext
DATA_EXPRESSIONS
LoadAnn('Y')  // load ANN Y into memory

RULE …

IF ASSE > 0.1
THEN TrainMomentum(100)  // train ANN in momentum and adaptive learning rate algorithm

//Following rule indicates that if ASSE has not been improved then increase the number of neurons
// by one and initialise the network and start training again.

RULE…

IF LastASSE < ASSE
THEN Hidden1 := Hidden1+1  // number of neurons of hidden layer1 is increased by 1
AND Initialise()  // initialise the network with current parameters
AND Train(100)  // start training the network

Fig. 19 Controlling ANN inside ES
```

```plaintext
Begin
ExpertInput ('Select C1,C2 from …')
ExpertOutput ('Select C3 from…')
RunExpert ('X')
AnnInput ('Select C1,C2 from …')
AnnOutput ('Select C4 from …')
RunAnn ('Y')
ExpertInput ('Select C3,C4 from …')
ExpertOutput ('Select C5 from …')
RunExpert('Z')
End
```

Fig. 20 The systems performing the same task by taking common inputs
Example 5
This example shows how a problem can be divided into subtasks, where a subtask can either be performed by ES or ANN. As shown in figure 21, ES X and ANN Y perform the different subtasks, where X taking input from C1, C2 and Y from C3, C4. The ES Z combines the result and produces output in column C7.

Example 6
This example is the same as that of example 5, but here both the systems share data and communicate to each other. As shown in figure 22, ES X and ANN Y perform the different subtasks, where X takes input from column C1 and C2 and Y from C3 and C4. The ES Z combines the result and produces output in column C10. X and Y communicate each other through columns C5 and C6 and share the column C7.

4.2 Operation in multi-user environment

In multi-user environment, HESS can be run in auto mode. There can be multiple instances of HESS running on the same machine or different machines. Each instance can be used to execute the systems or integrate the systems as described in examples through 1 to 6. Whenever, an instance of HESS starts, it can be set to start operation in auto mode. In auto mode, once the HESS is connected to the relational blackboard, it starts looking into the control table for any pending events. Other applications can be integrated with ES and ANNs. The control table is shared among many instances of HESS and other applications. The event table contains application specific hypotheses and their mappings with knowledge sources and events. Many KSs share data and communicate through relational blackboard.

The figure 23 shows the sequence of operations that are followed when a KSI posts a hypothesis in multi-user mode (shown in figure 24).
1. A KSI posts a hypothesis into hypothesis table.

2. The hypothesis trigger gets activated in response to posted hypothesis.

3. The hypothesis trigger checks whether posted hypothesis is linked hypothesis or not.

4. If the hypothesis is not linked then the hypothesis trigger looks into event table and puts corresponding KS/s and events into control table after applying application specific logic. The control goes to step 7. Otherwise, if the hypothesis is linked, the hypothesis trigger reduces the children count of parent hypothesis of posted hypothesis by one and makes the Parent_ID of posted hypothesis to nil. Making the Parent_ID to nil prevents the trigger from reducing the children count more than once for the same hypothesis.

5. The linked trigger is activated in response to update operation (when, the hypothesis trigger reduces the children count of a linked hypothesis by one) on linked table.

6. The linked trigger checks for any hypothesis whose children count has become to nil, if so, the trigger inserts the hypothesis into hypothesis table.

7. When a KSI is idle, it continuously looks into control table for any pending events.

8. If a KSI finds any pending event applicable to it, it performs some function/s in response to the event/s, and, in turn the function/s may post the hypothesis/hypotheses.

---

**Fig. 23 Steps followed when a hypothesis is posted**

- A KSI posts a hypothesis into hypothesis table.
- The hypothesis trigger gets activated in response to posted hypothesis.
- The hypothesis trigger checks whether posted hypothesis is linked hypothesis or not.
- If the hypothesis is not linked then the hypothesis trigger looks into event table and puts corresponding KS/s and events into control table after applying application specific logic. The control goes to step 7. Otherwise, if the hypothesis is linked, the hypothesis trigger reduces the children count of parent hypothesis of posted hypothesis by one and makes the Parent_ID of posted hypothesis to nil. Making the Parent_ID to nil prevents the trigger from reducing the children count more than once for the same hypothesis.
- The linked trigger is activated in response to update operation (when, the hypothesis trigger reduces the children count of a linked hypothesis by one) on linked table.
- The linked trigger checks for any hypothesis whose children count has become to nil, if so, the trigger inserts the hypothesis into hypothesis table.
- When a KSI is idle, it continuously looks into control table for any pending events.
- If a KSI finds any pending event applicable to it, it performs some function/s in response to the event/s, and, in turn the function/s may post the hypothesis/hypotheses.

---

**Fig. 24 The sequence of operations in a control cycle when a hypothesis is posted**
The knowledge sources other than HESS (i.e. ESs and ANNs) may respond to various user and control events. The HESS responds to the various predefined control events. The function PostHypothesis can be used in script to post a hypothesis. The database functions are used inside ES to explicitly post the hypothesis and look into control table to find out and process any pending events. Table 13 shows the list of predefined events to which an instance of HESS responds. The systems can be integrated in different integration modes that are illustrated through examples through 1 to 6 in multi-user mode also. Following examples illustrate how the systems can be integrated in multi-user environment.

Table 13: List of predefined events to which HESS responds

<table>
<thead>
<tr>
<th>Event</th>
<th>How HESS Responds</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_SCRIPT</td>
<td>Loads the script specified in KS_NAME column of control table into memory and executes it. If the script is already loaded, it is executed. The script remains loaded into memory until it is explicitly removed using EXIT_SCRIPT event. This helps to perform repetitive operations on the same script instead of loading it again and again. For example, once the ANN is loaded into memory for training, the training can be continued by sending TRAIN_NEXT event.</td>
</tr>
<tr>
<td>EXIT_SCRIPT</td>
<td>Removes the script from memory</td>
</tr>
<tr>
<td>TRAIN_NEXT</td>
<td>Continue the training of loaded ANN for next specified iterations</td>
</tr>
<tr>
<td>INIT_NETWORK</td>
<td>Initialise the loaded ANN with current parameters</td>
</tr>
<tr>
<td>SAVE_NETWORK</td>
<td>Saves the weights and other parameters of loaded ANN</td>
</tr>
</tbody>
</table>

Example 7

This example illustrates how the systems can be executed sequentially. In example 1, ES X and ANN Y were included in only one script and executed on a single instance of HESS. The scripts to execute X and Y can be separated so that both can be executed on different instances of HESS as shown in figure 25. The function PostHypothesis posts the hypothesis. If X and Y are to be executed sequentially, the Y should wait for X to complete execution. This information can be put into event table. The event table entries in table 14 show how hypotheses are mapped into knowledge sources and events. The execution of Y is mapped to hypothesis X Completed means that the execution of Y can not start until hypothesis X Completed is posted. The hypothesis Start Application maps to the KS that would start the execution of overall system, in this example X starts the execution. When Y completes the execution, the hypothesis Y completed would be posted which in turn would remove script X and script Y from memories of the corresponding HESS instances. All the events of KS TYPE S are responded by HESS because HESS executes the script specified in KS_NAME column.

Figure 26 shows the hypothesis trigger hyp_trigger that gets activated whenever a hypothesis is posted. It simply looks the newly posted hypothesis into the event table and schedules the corresponding KS/s and event/s for execution by placing them into the control table. For example, when X Completed hypothesis is posted, the hypothesis trigger hyp_trigger would move the entry that match X Completed from the event table into the control table.

(a): X Script

```plaintext
Begin
ExpertInput ('Select C1,C2 from …')
ExpertOutput ('Select C3,C4 from…')
RunExpert ('X')
PostHypothesis ('X Completed')
End
```

(b): Y Script

```plaintext
Begin
AnnInput ('Select C3,C4 from …')
AnnOutput ('Select C5 from…')
RunAnn ('Y')
PostHypothesis ('Y Completed')
End
```

Fig. 25 Scripts for example 7
Table 14: Event table for example 7

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>KS_NAME</th>
<th>KS_TYPE</th>
<th>EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Application</td>
<td>X</td>
<td>S</td>
<td>RUN_SCRIPT</td>
</tr>
<tr>
<td>X Completed</td>
<td>Y</td>
<td>S</td>
<td>RUN_SCRIPT</td>
</tr>
<tr>
<td>X Completed</td>
<td>X</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
<tr>
<td>Y Completed</td>
<td>Y</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
</tbody>
</table>

The `hyp_trigger` can be modified depending on the application needs. For example, suppose the execution of $Y$ is to be restricted on particular machine say $NIBM_1$, the trigger `hyp_trigger` can be modified to restrict the execution of $Y$ on machine $NIBM_1$ as shown in figure 27. The trigger checks whether hypothesis is $X$ Completed and accordingly puts $NIBM_1$ into the control table’s TERMINAL_ID column.

**Example 8**

This example illustrates *how the systems can be executed in parallel* on different instances of HESS. The ES $X$ and ANN $Y$ in example 5 can be executed in parallel and their results can be combined by $Z$. The scripts for $X, Y$ and $Z$ can be separated into three scripts as shown in figure 28. The event table entries and linked hypothesis table entries for this example are shown in table 15 and 16 respectively.

Since the ES $Z$ can not be started until ES $X$ and ANN $Y$ complete their execution. The hypothesis $X \textit{ and } Y$ Completed is a linked hypothesis not posted either by $X$ or $Y$. This hypothesis is linked to hypothesis $X$ Completed and $Y$ Completed. When both the hypotheses are posted, the linked trigger would post hypothesis $X$ and $Y$ Completed into hypothesis table. Figure 29 shows the linked_trigger that can be used to process the linked hypotheses. Figure 30 shows the modified `hyp_trigger` that checks whether a posted hypothesis is linked hypothesis or not.
Table 15: Event table for example 8

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>KS_NAME</th>
<th>KS_TYPE</th>
<th>EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Application</td>
<td>X</td>
<td>S</td>
<td>RUN_SCRIPT</td>
</tr>
<tr>
<td>Start Application</td>
<td>Y</td>
<td>S</td>
<td>RUN_SCRIPT</td>
</tr>
<tr>
<td>X Completed</td>
<td>X</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
<tr>
<td>Y Completed</td>
<td>Y</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
<tr>
<td>X and Y Completed</td>
<td>Z</td>
<td>S</td>
<td>RUN_SCRIPT</td>
</tr>
<tr>
<td>Z Completed</td>
<td>Z</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
</tbody>
</table>

Table 16: Linked_Hypothesis table for example 8

<table>
<thead>
<tr>
<th>ID</th>
<th>Hypothesis</th>
<th>Children</th>
<th>Parent_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X Completed</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Y Completed</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>X and Y Completed</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

```
Fig. 28 Scripts for example 8

(a) X Script
Begin
ExpertInput ('Select C1,C2 from…')
ExpertOutput ('Select C5 from … ')
RunExpert ('X')
PostHypothesis ('X Completed')
End

(b) Y Script
Begin
AnnInput ('Select C3,C4 from …')
AnnOutput ('Select C6 from …')
RunAnn ('Y')
PostHypothesis('Y Completed')
End

(c) Z Script
Begin
ExpertInput ('Select C5,C6 from …')
AnnOutput ('Select C7 from …')
RunExpert ('Z')
PostHypothesis ('Z Completed')
End
```

CREATE OR REPLACE TRIGGER linked_trigger
AFTER UPDATE ON linked_hypothesis FOR EACH ROW
BEGIN
/* if the children count becomes 0 (means all sub-ordinate hypotheses are posted), then entries of event table corresponding to hypothesis are inserted into control table */
IF :new.children=0 THEN
INSERT INTO control(ks_name,ks_type,event)
(SELECT ks_name,ks_type,event FROM event
WHERE hypothesis=:new.hypothesis);
END IF;
END;

Fig. 29 The Trigger linked_trigger on linked hypothesis table

In linked_trigger, instead of posting the hypothesis X and Y Completed, the trigger puts corresponding KS and event from event table into control table similar to that of hypothesis trigger. This because the mutating table error [10] occurs when the code to insert hypothesis into hypothesis table is written. This error occurs when a trigger attempts to modify or use the table being altered. The trigger hyp_trigger modifies linked_hypothesis table, upon which the trigger linked_trigger gets activated which can not insert into the table hypothesis.
because the trigger `hyp_trigger` would again get activated, creating a cyclic effect, and therefore, would cause mutating table error. The solution to this problem is to insert the corresponding KS/s and event/s into control table from event table, whenever the children count of a hypothesis becomes nil.

```sql
CREATE OR REPLACE TRIGGER hyp_trigger
AFTER INSERT ON hypothesis FOR EACH ROW
DECLARE
x INTEGER;
BEGIN
/*Check whether the hypothesis posted, is linked or not. If it is linked then variable x would contain the Parent_ID of linked hypothesis */
SELECT parent_id INTO x FROM linked_hypothesis WHERE hypothesis=:new.hypothesis;

/* If the hypothesis is linked, reduce the children count of parent hypothesis by one and set parent_id to 0 */
IF SQL%FOUND THEN
UPDATE linked_hypothesis
SET children=children-1
WHERE id=x;

UPDATE linked_hypothesis
SET parent_id=0
WHERE hypothesis=:new.hypothesis;
END IF;

/* if the posted hypothesis is not linked (i.e.not found in linked hypothesis table) then insert the corresponding KS/s and EVENT/s into the control table from event table */
EXCEPTION
WHEN NO_DATA_FOUND THEN
INSERT INTO control(ks_name,ks_type,event)
(SELECT ks_name,ks_type,event FROM event WHERE hypothesis=:new.hypothesis);
END;
```

Fig. 30 Modified trigger hyp_trigger to check the linked hypothesis

**Example 9**

This example shows how ANN training can be controlled and supervised by an ES, both may be running on different instances of HESS. Suppose an ES X controls the training of ANN Y. ES X gets or sets the ANN Y’s parameters through ANN table as shown in figure 31. Figure 32 show the scripts for this example. In the script Y, the function LoadAnn is used to load the ANN Y into memory. The hypothesis, Y Training Completed would be posted by the HESS, whenever, ANN Y completes specified number of training iterations. Table 17 shows the event table entries. ANN Y should be removed from the memory when the execution of ES X is terminated. Therefore, there are two entries against the hypothesis X Completed, the first, to terminate X, and, the second, to terminate Y. Figure 33 shows part of ES X code that controls the ANN. The database function `SELECTSQL` and `SQL` are used to get and set ANN parameters.

Fig. 31 Using and setting ANN Parameters
ANNs can be controlled and supervised from other applications similar to the way described in this example by setting or using ANN parameters through ANN table and control events.

**Example 10**

This example shows how HESS can be integrated to other application. In this example, ES and ANN are integrated to a Windows application *MYAPPL* developed in *Visual Basic* [31]. *MYAPPL* is used as a data-
The editing tool for ES CREDIT and ANN SINE as figure 34. The scripts that run ES Credit and ANN Sine are as shown in 35. MYAPPL shares two recordsets, Select * from Credit and Select * from Sine, with ES CREDIT and ANN SINE respectively. Whenever the data entry is completed or the modifications are completed, MYAPPL posts the hypothesis either Run Sine Network or Run Credit depending on the data currently editing.

![Image of MYAPPL interface](image)

Fig. 34 Application MYAPPL

As shown in table 18, the system starts with application MYAPPL. The MYAPPL does not start of its own, whenever any instance of HESS starts, it automatically executes any pending executable file/s. There is no event shown in table 18 against MYAPPL because MYAPPL starts execution with default function. Default procedure/function here means a function that is executed when application first starts. For example in C or C++, the system starts with main() function [26]. Once the MYAPPL starts, it itself manages the interaction with control table and posts hypotheses on its own. Figure 36 shows the partial execution and message flow between MYAPPL and HESS. Figure 37 illustrates how MYAPPL has its own control component, how it handles the events, and, how it posts the hypotheses. When MYAPPL starts, it connects to the database (variable DB is used as a database interface) and sets the time interval for control component. When the MYAPPL is idle, it looks for pending event/s after each time interval set.

![Image of script for MYAPPL](image)

(a) CREDIT

(b) SINE

Fig. 35 Scripts for example 10

There are two major user events: Clicking on Run Sine button and Run Credit button. In response to these event function/procedure RunSin_Click() and RunCredit_Click() are executed respectively. RunSin_Click() posts the hypothesis Run Sine Network while RunCredit_Click() posts the hypothesis Run Credit.
Table 18: Event table for example 10

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>KS_NAME</th>
<th>KS_TYPE</th>
<th>EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Application</td>
<td>MYAPPL</td>
<td>E</td>
<td>RUN_SCRIPT</td>
</tr>
<tr>
<td>Run Sine Network</td>
<td>SINE</td>
<td>S</td>
<td>RUN_SCRIPT</td>
</tr>
<tr>
<td>Run Credit</td>
<td>CREDIT</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
<tr>
<td>Close All</td>
<td>SINE</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
<tr>
<td>Close All</td>
<td>CREDIT</td>
<td>S</td>
<td>EXIT_SCRIPT</td>
</tr>
<tr>
<td>Sine Completed</td>
<td>MYAPPL</td>
<td>E</td>
<td>REFRESH_SINE_DATA</td>
</tr>
<tr>
<td>Credit Completed</td>
<td>MYAPPL</td>
<td>E</td>
<td>REFRESH_CREDIT_DATA</td>
</tr>
</tbody>
</table>

MYAPPL responds to two control events: REFRESH_SINE_DATA and REFRESH_EXPERT_DATA. These events are mapped to the hypothesis Sine Completed and Credit Completed as shown in table 18. The hypothesis Sine Completed is posted when ANN SINE completes execution, while hypothesis Credit Completed is posted when ES CREDIT completes execution. MYAPPL executes procedures RefreshSineData() and RefreshCreditData() in response to the control events REFRESH_SINE_DATA and REFRESH_CREDIT_DATA. These procedures refresh the recordsets with changed data.

Example 11

Instead of knowledge sources themselves posting the hypotheses, the triggers can be written that directly map the relational blackboard changes to the hypotheses or events. For example, in example 10, the hypothesis Sine Completed is posted whenever SINE ANN completes the execution. Instead of posting this hypothesis, the database trigger as shown in figure 38 can be written to post hypothesis into hypothesis table. This trigger is a statement level trigger and is on column $P$. A statement level trigger gets activated only once even if more than one record is affected while row level trigger gets activated for each record in affected records [1]. When the network completes execution, the output column $P$ of SINE table gets refreshed (with current output vectors). This activates the trigger Sine_trigger, which posts the hypothesis Sine Completed.

Example 12

This example shows how a query can be divided into sub-queries and how an ES or ANN can be shared and accessed in parallel by including them into different scripts. A query say select * from credit divided into two different independent sub-queries say select * from credit where ID < =100 and select * from credit where ID > 100. Instead of a single script as shown in figure 39(a), two separate scripts, say Credit1 and Credit2 can be written that share ES Credit but with different mutually exclusive or independent recordsets as shown in figure 39(b) and 39(c). These scripts can be executed in parallel on different instances of the HESS.
'The procedure PostHypothesis posts the hypothesis

Sub PostHypothesis(hyp As String)
    DB.Execute("Insert into hypothesis values('" & hyp & ")")
End Sub

'When user clicks on Run Credit Button, the hypothesis 'Run Credit' is posted

Private Sub RunCredit_Click()
    Call PostHypothesis("Run Credit ")
End Sub

'When user clicks on Run Sine Button, the hypothesis 'Run Sine Network' is posted

Private Sub RunSin_Click()
    Call PostHypothesis("Run Sine Network")
End Sub

'This is the control component/module, implemented using a timer

Private Sub Control_Timer()

'Looks for pending event for application, recordset RS stores the pending event/s

RS.Open "select event from control where ks_name='MYAPPL'", DB, adOpenStatic, adReadOnly
If Not RS.EOF Then  'if any pending event/s found
    'Remove all the events from control table that are applicable to MYAPPL
    DB.Execute("delete from control where ks_name='MYAPPL'")
    Control.Enabled = False  'Disable the timer (stop looking for events)
    Do While Not RS.EOF  'go through all the event/s
        Select Case RTrim(RS!event)  'execute appropriate action depending on event
            Case "REFRESH_SINE_DATA"
                Call RefreshSin
            Case "REFRESH_CREDIT_DATA"
                Call RefreshCreditData
        End Select
        RS.MoveNext
    Loop
    Control.Enabled = True  'Enable looking for any pending event/s again
End If
RS.Close
End Sub

'When user closes the application, the hypothesis 'Close All' is posted

Private Sub Form_Unload(Cancel As Integer)
    ...  
    Call PostHypothesis("Close All")
End Sub

Fig. 37 Partial Execution flow between HESS and MYAPPL
Discussion and Summary

The relational blackboard framework discussed in this paper represents a new and a different class of problem solving model that combines the client/server computing and blackboard model of problem solving to develop hybrid intelligent applications. The database coupling approach discussed in this paper relieves the hybrid system developer from addressing the problems of integrity, validity, consistency and security of the data, as these can be enforced centrally at underlying database system in multi-user environment. The database coupling provides the great degree of flexibility in selecting, preprocessing and manipulating the data before and after applying it to intelligent systems or exchanging it among the intelligent systems by using SQL-interface. The database coupling also makes it flexible to integrate the intelligent systems in different integration modes such as sequential, parallel or their combinations.

In the RBF, the knowledge sources are GUI applications that make the blackboard framework more interactive and user-friendly. The framework is modular and knowledge sources can easily be added or removed. The ESs and ANNs can be concurrently accessed and shared by multiple clients same as that of the information they access. The security constrains can be specified and imposed in the database for accessing the ESs and ANNs by multiple users. The knowledge sources themselves manage the interaction with the relational blackboard and look for the opportunity by referring to control table which stores the pending activities. This eliminates the need to have a separate central server component to control the knowledge sources once they are triggered or activated. There is no direct message exchange between knowledge sources; it is through the relational blackboard only. The ES is coupled to the database by setting up the order of input and output variables according to the linked recordset columns. Different recordsets can be linked externally to an ES. This eliminates the need to write the code inside the ES to open recordsets, navigate through them, assign column values to variables, update the recordset or writing SQL queries to update the data. In case of the ANN, the mapping is set implicitly between matrix I/O variables and a recordset whenever the recordset is opened and linked. The matrix I/O variables are automatically resized depending on the number of rows in the recordset.

The applications that require hybrid processing and blackboard model of problem solving can easily be developed using this kind of approach by using existing relational database management systems instead of developing and implementing the integration frameworks from scratch.

---

Fig. 38 Trigger Sine_trigger

```sql
/* This trigger is not row level but a statement level trigger. The column P stores the output vectors */
CREATE OR REPLACE Sine_trigger
AFTER UPDATE OF p ON Sine
BEGIN
  INSERT INTO hypothesis VALUES ('Sine Completed');
END;
```

Fig. 39 Scripts for example 12

(a) Credit

```
Begin
  ExpertInput('Select * from Credit')
  RunExpert('Credit')
End
```

(b) Credit1

```
Begin
  ExpertInput('Select * from Credit where ID<=100')
  RunExpert('Credit')
End
```

(c) Credit2

```
Begin
  ExpertInput('Select * from Credit where ID>100')
  RunExpert('Credit')
End
```

---
References


[19] MS-Excel is a spreadsheet package from Micro-Soft Corporation and is a part of MS-Office.


[31] *Visual Basic* is a programming language of MicroSoft Corporation under Windows environment.

[32] *Visual C++* is a Microsoft's C++ programming language for Windows environment.

[33] *Windows 95*: an operating system from MicroSoft Corporation.


**Footnotes**

1 ODBC is an interface for connecting heterogeneous databases proposed by Microsoft Corp. and supported by number of database vendors.

2 ADO is Microsoft’s strategic, high-level interface to all kinds of data, provides consistent, high-performance access to data, for, a front-end database client or middle-tier business object using an application, tool, language, or even an Internet browser.

3 A control used to control recurring events in an application. The timer is not visible at run time.

4 Visual Basic for Applications (VBA) is a single, common application scripting language and environment that users and developers can leverage across all their Windows desktop. Visual Basic for Applications is included in Microsoft Office and other Microsoft applications.