Knowledge-based systems in the commercial environment

by E. D. Hodil
C. W. Butler
G. L. Richardson

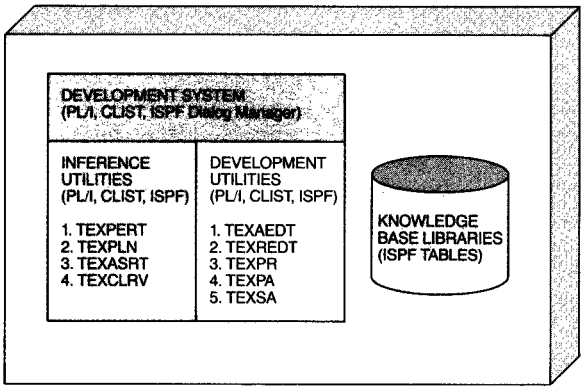
Knowledge-based systems are among the first applications of artificial intelligence to make the crossover from the laboratory to the real-world commercial environment. Typically, artificial intelligence systems have been implemented in the LISP programming language on specialized hardware. The experimental nature of early systems has allowed many of them the luxury of having little or no interface to existing hardware, software, or data. In this paper, arguments are presented to demonstrate the feasibility of implementing knowledge-based systems using traditional hardware and software. Also, an architecture is proposed for knowledge-based shell systems that is compatible with the software development environment of large commercial information systems organizations. To demonstrate these concepts, an example system is shown.

During the past several years, the knowledge-based system (KBS) has emerged as a new class of software. Based upon theoretical concepts conceived in the early 1950s in various academic institutions such as the Massachusetts Institute of Technology, Carnegie Institute of Technology, and Stanford University, these systems are finding their way out of the laboratory and into the business place. If the prognosticators are correct, they will alter the way in which computers are used in the years to come. Knowledge-based systems are designed to emulate narrowly defined subsets of human decision making. Unlike traditional transaction and algorithmic systems, these systems can operate under conditions of incomplete and uncertain information. They may also deal with "fuzzy" data such as opinions or qualitative evaluation. Moreover, in the process of reaching some operational goal, KBS logic is generally self-diagnostic. On command, a KBS will explain the inferences used to reach conclusions. Because they provide a new approach to the process of decision making, it is easy to see why KBSs are projected to occupy a vital role in future information systems architecture.

In concept, KBSs would appear to hold the ideal solution for a multitude of problems encountered in the commercial information systems world. Surprisingly, real progress has been slow in coming. To date, fewer than one hundred KBSs are known to be in commercial use. According to conventional wisdom, the shortage of real-world systems can be explained by the newness of the technology and the lack of specially trained personnel. A more likely explanation is that KBS shell system developers have, for the most part, opted for languages (LISP and PROLOG) and specialized hardware (single-user LISP workstations) that inherently isolate potential business users from both the existing operational architecture and the associated base of trained personnel.

There are sound technical and economic reasons for building KBSs within the architecture of a commercial information systems environment. Most significantly, the training investment for computer professionals makes the use of existing traditional tools

Figure 1 TEXpert KB Development System



essential wherever possible. With use of existing professional skills and conventional development concepts, much of the complex implementation process can be subjugated. Another reason justifying implementation using traditional developmental tools is the fundamental issue of system integration. A stand-alone special-purpose environment creates a new architecture and thereby introduces serious constraints to network compatibility and interconnectivity. Despite the fact that KBS technology does have certain characteristics which make it unique, its components, particularly the data acquisition subsystem, may need linkage to the existing information systems architecture. As the KBS user community grows, it is becoming more obvious that the KBS should be integrated with the existing information system network and its current desktop workstation. Thus, the ergonomic factors also favor techniques for merging knowledge-based systems into the mainstream of commercial data processing.

There are several precedents which refute the argument that all KBS applications, or for that matter all artificial intelligence (AI) applications, require special-purpose languages or hardware. For example, INTELLECT,[1] a commercially available natural-language data base query program, is implemented in PL/I and runs on various IBM mainframes. INTELLECT uses several of the major data base management systems. A more recent program, the IBM Expert System Environment/VM (ESE) program product,[2] written in Pascal, demonstrates that KBSs may also be developed using procedural languages and conventional hardware. ESE also features connectability to external data sources and external programs. Prior to ESE, Weiss and Kulikowski constructed several knowledge-based systems using a shell system called EXPERT which is written in FORTRAN.[3]

Although it seems clear that applications can be developed today using tools and personnel that are already in place, it is important that the insular nature of special workstations and languages not be carried over to mainframe/traditional language implementations. Historically, one of the most successful paradigms for integrating new functions into commercial information systems has been the "programmer's toolbox" approach. Here, the desired function is decomposed into a library of single-purpose subroutines, each of which may be used to build customized applications. The chief advantages of this method are that (1) it generally does not require the application developer to learn new programming languages; (2) it allows programmers to build customized systems which use only needed subfunctions; and (3) it permits the "orchestration" of many types of functions (i.e., graphics, screen management, data base management, knowledge-base management).

To support this position, the architecture of a shell system written primarily in PL/I is outlined, and then a sample application developed using the shell is discussed. Readers who are unfamiliar with the general features of knowledge-based systems are referred to Hayes-Roth et al.[4] and Weiss and Kulikowski[3] among others.

TEXpert

TEXpert is a shell system written in PL/I and IBM's Interactive System Productivity Facility (ISPF). Since ISPF supports menu-driven applications, it combines user-friendly development and consultation facilities with a discrete set of programmer's toolbox utilities. The guiding requirements for the construction of TEXpert were threefold:

1. To engender the ease-of-use features of most stand-alone workstation tools, i.e., easy-to-use knowledge-base editors and debuggers.
2. To connect easily with existing software and data bases.
3. To implement a development interface that programmers, analysts, and users of IBM MVS/TSO environments would recognize and quickly learn. Each of these groups needs to access the system at its own skill level.

To accomplish these objectives, the KBS function was first decomposed into the toolbox utility programs.

Second, a development system, depicted in Figure 1, was built to satisfy end-user requirements and those of the technical staff. At the core of the development system are the inference utilities of the toolbox. Editing and testing tasks associated with knowledge-base construction are handled by the development utilities. The development utilities also encompass features such as printing and static analysis. The driver program of the development system gives the user a structured interface to both the inference and the development subsystems. Physically separate from the procedural pieces of the system are the knowledge bases which contain application-specific knowledge and data. In the following sections, each of these components is discussed.

Central to any KBS is the knowledge base.

Knowledge-base architecture. Central to any KBS is the knowledge base. Attributes (object properties) and rules (expert heuristics) are the primary data structures of a knowledge base. Unique to TEXpert is the use of ISPF tables as the repository for attribute and rule data. The ISPF product is itself a good example of the toolbox utility approach advocated here. Each knowledge base is implemented as an ISPF table library with internal representations of attributes and rules stored separately in two members of the partitioned dataset of the tables.

Rules in the TEXpert system have the usual "if" and "then" parts of production rule systems. Rules are typically entered using the rule editor of the development utility but can also be created by application-specific procedures. Internally, rules are stored in a compressed text format to minimize the amount of run-time parsing. The "if" part, or antecedent of a rule, contains attribute value comparisons that must be "true" for the rule to "fire" (i.e., assert its consequent clauses). Antecedent clauses are formed using an operator prefix notation, another parsing optimization technique. The allowed comparison operators are the usual EQ, NE, GE, GT, LT, and other operators. The clauses may be conjoined and/or disjoined. Consequent clauses of the "then" part of the rule are used to assign values for attributes when and if a rule "fires." Each consequent clause has an *a priori* certainty factor associated with it. The certainty factor may take a value from zero to one, where one is absolute certainty.

Attributes are stored with definition data and, where appropriate, with pointers to rules that reference them. Attributes may be declared as one of several data types:

STRING—character string up to 31 characters
INTEGER—16-bit integer
LONG—32-bit integer
FLOAT—16-bit floating point
DOUBLE—32-bit floating point
ENUM—ordered list of values

The ENUM data type is similar to the Pascal enumerated data type. It allows the user to define alternative collating sequences for use in rule comparisons. For example, a user may declare DAY_OF_WEEK as an ENUM attribute containing a possible value from SUNDAY to SATURDAY. Attributes may also be defined to take only one value or to take a set of values. Finally, the user may specify the order of sourcing for the value of an attribute (discussed in the next section) and may declare auxiliary text data for use in querying the user.

Admittedly, the TEXpert shell lacks sophistication in terms of knowledge representation and optimal performance. One planned functional enhancement to the system is the addition of explicit objects. Currently, with only one implied object allowed, the user must embed object references within attributes. Embedding objects can result in overly large knowledge bases and a loss of generality. For instance, the expression

```
EQ COLOR_OF(X)WHITE
```

where "X" is an object variable, can be used in many contexts, whereas the current TEXpert expression

```
EQ COLOR_OF_DOG WHITE
```

may only be used in the context of "dogs."

The performance issue could easily be addressed with the use of text string encoding. As yet, this has not proved necessary since ISPF table services provide data compression, and TEXpert has not exhibited serious performance degradation. In addition, some performance improvement has already been ob-

Figure 2 ASKUSER screen

```
---------------------- T E X P E R T ------------ROW 1 OF 2
COMMAND ===>                                  SCROLL ===> CSR

PF13  =HELP   ENTER =DON'T KNOW   PF04  =WHY   PF05  =RULE
PF19  =UP     PF20  =DOWN         PF15  =END

LOCATION

 LOOK AT A MAP OF THE UNITED STATES, AND DETERMINE
 YOUR POSITION. NOW, IS THIS LOCATION:
 (CHOOSE ONE OF THE FOLLOWING)



=== OPTION ============== COMMENT ========================
    WEST_OF_MISSISSIPPI   ?
    EAST_OF_MISSISSIPPI   ?
********************** BOTTOM OF DATA **********************
```

tained by storing precompiled pointers to relevant attributes along with the rules. These pointers are used by TEXpert utilities to help optimize searches.

All in all, the table structure of rule and attribute data lends itself to an ISPF table services implementation. This system works well except for limited table-sharing facilities in WRITE mode. There also needs to be an improved level of security for sensitive data. Alternatives to these design limitations are currently under consideration. One primary approach to addressing these items is to use IBM's DB2 relational data base for rule storage.

Inference utilities. The fundamental programs upon which the development system is built are the inference utilities. Four programs perform the functions traditionally associated with the inference engine of stand-alone systems, e.g., problem space search, pattern matching, user query, and explanation. The four primary inference utilities are

1. TEXPERT—a backward-chaining, best-first-search inference utility
2. TEXPLN—a program that "follows" the line of inferences and produces a tabular representation for the hierarchy of successful rules
3. TEXASRT—a program for asserting (and retracting) attribute values
4. TEXCLRV—a program to refresh the knowledge base with null values

The TEXPERT utility is invoked with arguments indicating the appropriate knowledge base to be explored for a specific problem area. The user is additionally able to specify a sensitivity factor (ranging between zero and one) which is used as the criterion for rule firing. Rules for which the cumulative evidence exceeds an assigned threshold are "fired," and their consequents are made a part of the knowledge base.

TEXPERT propagates certainty through multiple levels of rules in the same manner as EMYCIN.[4] If the antecedent of the rule is composed of conjuncts (statements connected by the AND operator), the maximum certainty factor of the conjuncts is multiplied with the *a priori* certainty factor of the consequent. For disjunctive (OR) antecedents, the minimum certainty factor is used. Although this scheme is far from perfect, it is one that is commonly employed. It is important to note here that TEXPERT certainty factors do not necessarily represent pure statistical probability.

The TEXPERT utility accepts data from a wide range of external sources limited only by the imagination of the KBS builder. The built-in data acquisition method is the ASKUSER facility. An example ASKUSER prompt panel is shown in Figure 2. Note that unlike the user queries of many shell systems, ASKUSER provides a range of values from which the user may select. When the query is constructed in this way, the user is not required to tediously answer many questions of the form "Is it true that attribute X has value Y?" The self-diagnostic "Why" and "Rule" commands that are typically found in knowledge-based systems are also included in ASKUSER. The "Why" command shows the user the stack of rules under consideration, and the "Rule" command shows the "pretty-printed" text of the rule that is under immediate scrutiny.

It should be pointed out that for many applications the ASKUSER facility will not be a satisfactory method of data acquisition. As mentioned earlier, such systems as existing EDP (electronic data processing), MIS (management information system), and DSS (decision support system) contain data upon which knowledge engineers of future applications will undoubtedly want to draw. It is also true that customized user interfaces will be needed for some systems. To give the knowledge-base builder the ability to utilize external data and simultaneously allow for customized interfaces, the TEXPERT program contains a built-in method for implementing user-defined exit routines. Synchronous exits from the shell system are defined to TEXPERT in a run-time argu-

Figure 3 TEXPLN screen

```
---------------------------- T E X P E R T ----------------ROW 1 OF 12
COMMAND ===>                                                SCROLL ===> CSR

PF15  =END    PF19  =UP    PF20  =DOWN

THE VALUE FOR SOCK_COLOR IS BROWN.
 (CERTAINTY FACTOR = 0.34)
THE PATH OF REASONING IS AS FOLLOWS:
(YOU MAY SELECT A RULE TO SEE ITS TEXT)

=== LINE NO. ================== SOURCE ==============================
       1.     BROWN RULE.
       2.       BUSINESS RULE.
       3.         YOU SAID JOB = PROGRAMMER
       4.         YOU SAID JOB = PROGRAMMER
       5.         YOU SAID JOB = PROGRAMMER
       6.       ACTIV RULE.
       7.         BUSINESS RULE.
       8.           YOU SAID JOB = PROGRAMMER
       9.           YOU SAID JOB = PROGRAMMER
      10.           YOU SAID JOB = PROGRAMMER
      11.         YOU SAID MARITAL_STATUS = MARRIED
      12.       YOU SAID WEEKDAY = TUESDAY
*************************** BOTTOM OF DATA ****************************
```

ment which consists of an array of subroutine entry points. Associated with each entry address is the character representation of the name of the subroutine. This exit array is used by TEXPERT (via a table lookup and subscripted CALL) whenever the services of an exit routine are required. All data communication between TEXPERT and exit routines is accomplished through the TEXASRT utility.

The inference mechanism of TEXPERT deviates a bit from the norm as a result of lessons learned from building previous KBSs. One extremely useful feature is the variable sourcing sequence for attributes. As TEXPERT attempts to resolve a value for an attribute, it first retrieves a sourcing sequence record from the knowledge base. The sourcing sequence record indicates the order in which various sources of data will be tried. The sourcing sequence of a typical attribute may specify that rules be tried first, followed by an exit routine. Failing these, the knowledge-base designer may indicate that a default value be assigned to the attribute, or that a query be posed to the user. As in IBM's ESE, alternative sourcing sequences may be programmed by the knowledge-base designer using the development system. Optionally, the source sequencing may be dynamically composed using the exit facility.

Another inference feature that deserves mention is the best first search. When considering alternative rules to try, TEXPERT will order the relevant rules so that rules with the highest *a priori* certainty factors will be tried first. Thus, a knowledge-base designer may control the order in which rules are tried by

Figure 4 TEXAEDT edit options

```
EDIT ---------- T E X P E R T ----------
OPTION ===>

   BLANK  - DISPLAY ATTRIBUTE LIST

   D      - EDIT DEFINITION, ALIAS, SOURCE ORDER
   A      - EDIT ASKUSER SCREEN
   V      - EDIT VALUE LIST
   E      - EDIT EXIT LIST

   DEL    - DELETE ALL DATA FOR ONE ATTRIBUTE
   CLR    - CLEAR ALL VALUES FOR ALL ATTRIBUTES

KB LIBRARY:
   PROJECT        ===> AK45EDH
   KNOWLEDGE BASE ===> PROGEVAL
   ATTRIBUTE      ===>
```

Figure 5 TEXREDT rule edit

```
EDIT ---------- T E X P E R T ----------
COMMAND ===>            SCROLL ===> CSR
****** ******* TOP OF DATA *************
000001  IF
000002     AND
000003        EQ CAREER BUSINESS
000004        EQ SOCIAL_ACTIVITY ACTIVE
000005        GT WEEKDAY SUNDAY
000006     ENDA
000007  THEN
000008     ASSERT
000009        SOCK_COLOR BROWN 0.45
000010     ENDA
****** *************** BOTTOM OF DATA ********************
```

appropriately manipulating the certainty factors in rule consequents.

The TEXPLN utility uses pointers built by TEXPERT to generate an indented table, illustrated in Figure 3, that shows the path of rules used to derive attribute values. Like TEXPERT, it also accepts arguments for the name of the knowledge base and the goal attribute. The recursive mechanism of TEXPLN begins by determining the source for the goal attribute. Sources for attribute values are stored whenever the TEXPERT program asserts a value for that attribute. Some sources such as user exit routines and the user's terminal input are "leaf" nodes in the explanation "tree." Rule sources, however, require more elaboration. When applicable, the TEXPLN program extracts the relevant rule, and in a manner very similar to that of TEXPERT's backchaining mechanism, recursively develops explanation subtrees for each clause of the antecedent. The user may specify whether or not redundant paths are to be shown.

The TEXASRT subroutine gives the customized KBS builder an easy-to-use and well-defined interface to knowledge-base data structures. TEXASRT is the same routine that is used from within TEXPERT to add or assert values for attributes. Besides attribute values, TEXASRT also updates the source pointers (used by TEXPLN) and the associated certainty factors for values.

The single function of the TEXCLRV subroutine is to delete the value records from the knowledge base. This function is used between invocations of the TEXPERT program. If TEXCLRV is not used before calling TEXPERT, all previous values will remain in effect. TEXCLRV also clears the source pointers and the certainty factors for values.

Development utilities. The five programs of the development utilities can be functionally classified as editing or maintenance programs. Specifically, these procedures are

1. TEXAEDT—builds and edits knowledge-base attributes
2. TEXREDT—builds and edits knowledge-base rules
3. TEXPR—archives rules
4. TEXPA—archives attributes
5. TEXSA—performs static analysis

Like the inference utilities, these subprograms are also available for use in custom applications, though experience to date has shown that they are used to a lesser degree.

There are two paradigms for knowledge-base editors: batch compilation and simultaneous edit and compile. With use of the compiler scheme, a language for declaring attributes and rules is used to define a knowledge base. The user builds the knowledge base using a standard text editor and "compiles" it using the development program compiler. The Knowledge Engineering System (KES) shell system[5] is constructed in this way.

The alternative method, employed by TEXpert, utilizes specialized edit facilities which simultaneously compile the source text. Figure 4 shows one of the

formatted panels used to edit attributes. Processing behind the panel checks the input and compiles it when a save command is issued. Similarly, an example rule edit is shown in Figure 5. The underlying function parses the rule both syntactically and globally and performs other edit support tasks such as "pretty printing" of the rules and copying rule source text from other ISPF tables. A specialized edit and compile function is superior because it minimizes the time required to complete a single iteration of the create-compile-debug-edit cycle. A drawback is that users are required to learn slightly new editing facilities, even though this factor has been minimized by building an interface that closely resembles the typical ISPF environment.

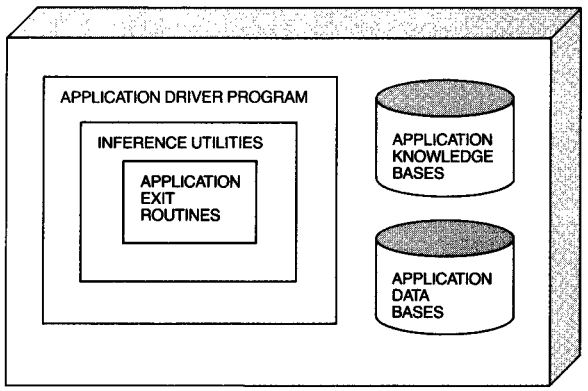
The TEXPR and TEXPA knowledge-base maintenance programs are used to format printed copies of rules and attributes. Both TEXPR and TEXPA allow the user to print individual rules and attributes or the entire rule/attribute set for an application. The static analysis program, TEXSA, provides tabular overviews of knowledge bases. One view shows the relationships among attributes in the attribute hierarchy. Another perspective shows the interconnections among the rules of a knowledge base. The rule interconnection graph is very similar to the TEXPLN graph shown earlier. The output of TEXSA differs from that of TEXPLN in that TEXSA attempts to show the case where all rules have fired. Taken together, these two reports can be used to identify potential rule inconsistencies and "dangling ends" (branches of rule and attribute trees that are disconnected from the trunk). This is a new form of program debugging, for which firm methodology has yet to be developed.

The development system. A builder of a KBS (either a programmer or an end user) can access the services of the development and inference utilities via the development system. The system is a collection of ISPF panels and Time Sharing Option (TSO) command lists designed to guide the user through the construction of a knowledge-based system. Figure 6 shows the primary options menu for the development system of TEXpert. Note that there are options for creating, editing, testing, and deleting knowledge bases. The test option is implemented with the same TEXPERT and TEXPLN subprograms previously described. Thus, consultations work exactly the same in finished production systems as they do in the development phase. It is also noteworthy that no special debugging facility is required or supplied, since the inherent "Why" and "Rule" capabilities of the TEXPERT program fulfill this role.

Figure 6 TEXPERT primary options

```
---------------------- T E X P E R T ----------------------
OPTION  ===>

   0  CREATE KB        - ALLOCATE KB DATASETS
   1  EDIT ATTRIBUTES  - ADD/CHANGE/DELETE ATTRIBUTES
   2  EDIT RULES       - ADD/CHANGE/DELETE RULES
   3  TEST KB          - TEST KNOWLEDGE BASE
   4  DELETE KB        - DELETE KB DATASETS
   5  UTILITIES        - KB DEVELOPMENT UTILITIES
   T  TUTORIAL         - DISPLAY INFORMATION ABOUT TEXPERT
   X  EXIT             - EXIT TEXPERT

ENTER END COMMAND TO TERMINATE TEXPERT.

                        USERID - AK45EDH     TERMINAL - 3278
                        TIME   - 13:03       PF KEYS  - 24

                                             RELEASE  - 1.0
```

Figure 7 Architecture of a custom KBS application

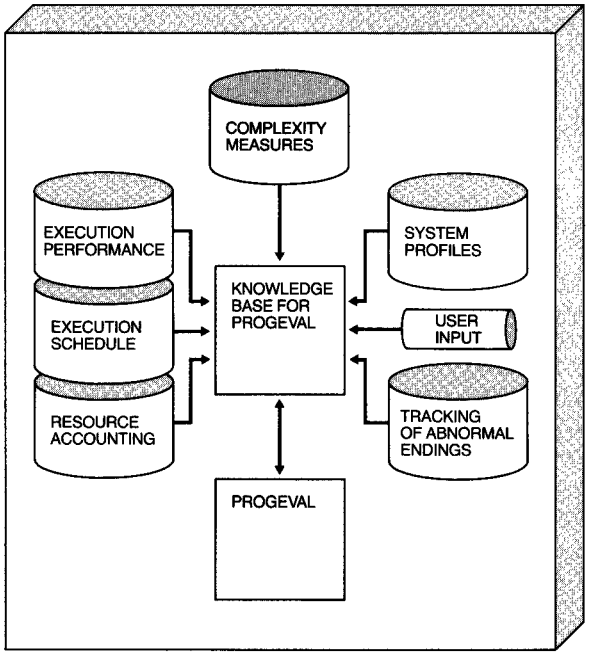


A final item addressed in the development system is the integrated on-line tutorial. Most shell systems have "help" functions, and TEXpert is no different. The TEXpert tutorial subsystem follows the well-established ISPF tutorial interface with which a large body of professionals are already familiar. Moreover, this facility is integrated into the ASKUSER function of the TEXPERT program, thereby allowing application-specific tutorial/help subsystems to be included easily in custom applications. Users have the option of entering the tutorial directly from the main menu, or invoking it via the help program function key where needed.

A customized application

The form of a custom KBS application built using TEXpert is shown in Figure 7. A customized appli-

Figure 8 PROGEVAL data sources



cation driver program, similar to that of the development system, is designed and implemented by the knowledge-base designer. CALL invocations are made to required inference TEXpert utilities. There is no limit to the number of times an inference utility can be invoked; however, recursive invocations of the TEXPERT program currently are undefined. Optionally, the developer may specify application exit routines through the addresses passed to the TEXPERT program. Logically, though not lexically, the exit routines are subroutines of the inference utilities.

The ISPF tables containing the knowledge base and any data files required by the application must be allocated before the services of the inference utilities are requested. Also, the required panel library allocations must be made before ISPF is entered, or afterward via the ISPF LIBDEF facility.

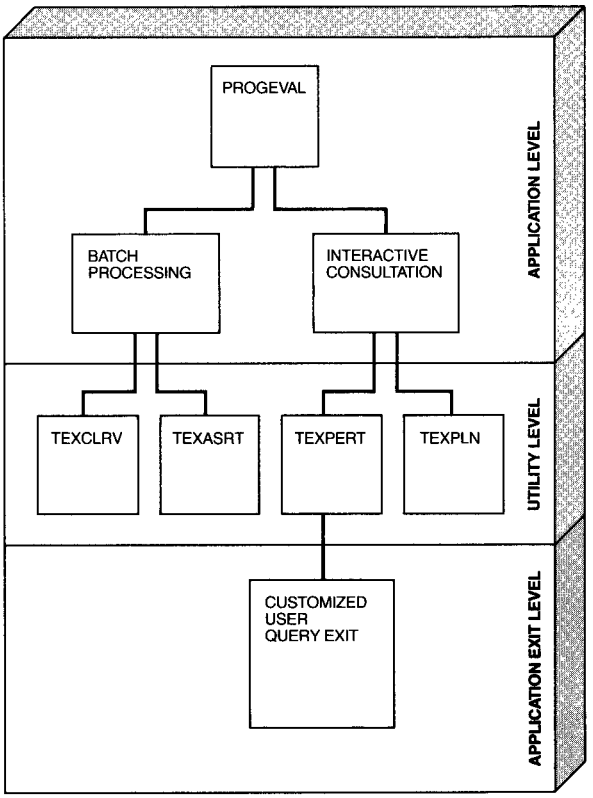
PROGEVAL: A sample system. The PROGEVAL prototype application is used to evaluate application programs and prescribe maintenance activities. This system, which is still under development, consists of two parts:

1. Program evaluation—rules and data to determine program "fitness"
2. Prescription—rules to determine what maintenance action is required to fix a code unit

The evaluation process is based upon measured parameters from four areas: source module complexity, program run-time failures, job control language failures, and operations (manual procedure) failures. The prescriptive half of the system heavily overlaps evaluation. Most of the attributes are, in fact, the same. Factors that are unique to prescription are the operating cost for the maintenance of the program and intangibles such as political sensitivity. Figure 8 shows the relationships between PROGEVAL and the various data sources. An important feature of PROGEVAL is its reliance on external data sources other than user consultation. Except for subjective evaluations concerning the quality of a code module and the political climate surrounding the parent system of the program, all data are gathered from existing tracking systems.

Constructing PROGEVAL. The PROGEVAL system was constructed using the TEXpert architecture. A primary design goal was to construct the application in a manner consistent with ISPF convention. Thus, many of the features expected by ISPF users, such as

Figure 9 PROGEVAL module hierarchy



hierarchical menus, data entry panels, and informational messages, appear in the user interface.

Figure 9 illustrates the dual-subsystem nature of this application. Note that the system consists of a batch portion and an on-line consultation part. The purpose of the batch component is to gather data from the external sources discussed earlier and to load the results into an ISPF table. The batch subsystem is not executed each session because the data are generally not refreshed on a day-to-day basis. The on-line consultation segment serves to gather user input and to present prescriptions regarding the status of production load modules. The following subsections explain each component of the design in more detail.

Batch component. Each of the external systems schematically pictured in Figure 8 contains facts that are needed by the knowledge base. PROGEVAL draws data from these external systems for use in the on-line consultation. In doing so, the batch subsystem uses the TEXCLRV program to remove previous data from the knowledge base, and then the TEXASRT facility assigns current values for attributes. The batch updating interface is initiated when the user selects option 1 on the primary menu shown in Figure 10. Execution of this option may be periodic since the data change infrequently.

On-line component. The on-line consultation system is initiated when the user selects option 2 of the PROGEVAL primary options menu. After invocation, the user is presented with a program specification screen, as shown in Figure 11. From this screen it is possible to further specify the application library (the set of programs the user is responsible for) and the particular program to be examined by the knowledge-based system.

Alternatively, the user may leave the program specification field blank and be presented with a selectable screen as shown in Figure 12. Here, a scrollable list of programs is displayed along with the date of the most recent update of the supporting knowledge base (from the batch subsystem). The user may then choose to display the supporting knowledge base, by entering the letter "D," or he may begin the consultation by entering the letter "S" next to the desired program.

Before the TEXPERT inference utility is invoked, the user is prompted to specify the level of confidence required in evaluating the problem program (see Figure 11). This number is then converted to the appropriate sensitivity factor. As stated earlier, the

Figure 10 PROGEVAL primary options

```
--------------------- P R O G E V A L ---------------------
OPTION  ===>

   1  COLLECT DATA     - SUBMIT DATA COLLECTION JOB
   2  CONSULT          - ON-LINE CONSULTATION
   T  TUTORIAL         - DISPLAY INFORMATION ABOUT PROGEVAL
   X  EXIT             - EXIT PROGEVAL

ENTER END COMMAND TO TERMINATE PROGEVAL

                        USERID - AI45EDH     TERMINAL - 3278
                        TIME   - 10:25       PF KEYS  - 24
```

Figure 11 PROGEVAL program specification screen

```
CONSULT ------------- P R O G E V A L ---------------------
COMMAND ===>

PROGRAM LIBRARY:
   PROJECT  ===> AI45EDH
   LIBRARY  ===>
   PROGRAM  ===>          (BLANK FOR PROGRAM SELECTION LIST)

CONFIDENCE LEVEL  ===>   (0 TO 100)
```

Figure 12 PROGEVAL program list

```
CONSULT ---------- P R O G E V A L -------------ROW 1 OF 11
COMMAND ===>                                  SCROLL ===> CSR
--- PROGRAM ---- CREATED ------ MODIFIED -------- USER ----
    AI450001      09/25/85       11/15/85          AI45EDH
    AI450002      09/13/85       11/15/85          AI45EDH
    AI450003      09/22/85       11/15/85          AI45EDH
    AI450004      09/25/85       11/15/85          AI45EDH
    AI450005      09/25/85       11/15/85          AI45EDH
    AI450006      09/13/85       11/15/85          AI45EDH
    AI450007      09/13/85       11/15/85          AI45EDH
    AI450008      09/25/85       11/15/85          AI45EDH
    AI450009      09/25/85       11/15/85          AI45EDH
    AI450010      09/22/85       11/15/85          AI45EDH
    AI450011      09/25/85       11/15/85          AI45EDH
********************* BOTTOM OF DATA **********************
```

Figure 13 CALLS to TEXPERT and TEXPLN

```
/*********************************************************************/
/**                  M A I N   P R O C E S S I N G                  **/
/*********************************************************************/
 POS = INDEX(PARMS,',');                          /* FIND FIRST ',' */
 DO I = 1 TO 10 WHILE (POS ¬= 0);
    PARM(I) = SUBSTR(PARMS,1,(POS-1));            /* GET 1ST PARM   */
    PARMS = SUBSTR(PARMS,(POS+1));                /* MAKE PARMS THE */
    POS = INDEX(PARMS,',');                       /*  REMAINDER     */
 END;
 PARM(I) = PARMS;                                 /* GET LAST PARM  */

 KBNAME = PARM(1);                                /* ASSIGN PARMS   */
 TARGET_ATTR = PARM(2);                           /* FOR CALLS      */
 CSENS = PARM(3); PSENS = CSENS; SENSITIVITY = ROUND((PSENS/100),2);
 CEXPL = PARM(4);
 IF CEXPL = 'YES' THEN
    EXPLAIN = '1'B;
 ELSE
    EXPLAIN = '0'B;

                                                  /* CALL KB RTNS   */
 CALL TEXPERT(KBNAME,TARGET_ATTR,SENSITIVITY,EXPLAIN,EXIT);
 IF EXPLAIN THEN
    CALL TEXPLN(KBNAME,TARGET_ATTR);
 RETURN; /* TO SYSTEM */
```

Figure 14 PROGEVAL customized user query

```
CONSULT ------------ P R O G E V A L ---------------------
COMMAND ===>

PF03  =END   PF04  =RULE   PF05  =WHY

VISIBILITY IS MEASURED BY THE LEVEL OF MANAGEMENT THAT WILL
VIEW THIS PROGRAM'S OUTPUT, OR THAT WILL BE DIRECTLY
AFFECTED BY IT.

PREVIOUS RESPONSE = HIGH

CURRENT RESPONSE ===> L    (H IGH, M EDIUM, L OW)
```

sensitivity factor is used by the inference engine to decide whether to accept the consequents of a rule. An excerpt from the control program of the on-line subsystem (with calls to TEXPERT and TEXPLN) is shown in Figure 13.

The built-in ASKUSER routine of TEXpert was not used in this application, since the special requirement of showing past user responses is necessary. The customized panel shown in Figure 14 is displayed by an exit subroutine associated with the VISIBILITY attribute.

The usual explanation routines (with a slight modification to handle dual-role graphs) are used to display the conclusions and prescriptions of the PROGEVAL consultation. At this point the user is returned

to the program specification screen, and the process iterates until the user exits the program.

Concluding remarks

In this paper, we have examined the rationale for developing knowledge-based systems within the traditional framework of commercial information systems. This is in contrast to much of the literature, which suggests that unique machines and languages are prerequisites for KBS implementation. Second, we have demonstrated the utility of a multilayered architecture for shell systems. Within this architecture, the diverse skill levels of many professionals can be applied without shackling application programmers to restricted, narrow interfaces.

A shell development environment written in a traditional procedural language and using common system software offers numerous advantages.

A shell development environment written in a traditional procedural language and using common system software offers numerous advantages. The TEXpert approach outlined here allows knowledge-based systems to be viewed as little more than the implementation of a new application, a task which professional programmers are accustomed to doing. A second significant advantage is its natural integration into the existing information systems architecture. Third, the TEXpert architecture combines a flexible building block approach which can be used to handle a wide variety of custom knowledge-based application needs. Consequently, the product can be delivered to the client through the same standard terminal network without special hardware. More significantly, the existing inventory of software tools, including graphics, data bases, editors, and debuggers, is readily at hand to facilitate implementation. Each of these features assists in the orderly blending of knowledge-based systems into the existing operational environment.

As knowledge-based systems mature in the organization and productive applications evolve, two developments will strongly favor the approach outlined here. First, future applications will need to tap the existing installed operational data bases extensively in order to feed their inferencing processes. Second, the user community will be widespread across the organization, mandating systems that fit into the existing "way of doing things." Data base access and telecommunications compatibility are vital issues in the future KBS design architecture. Nowhere is technology better defined than in the traditional transaction environment using the type of tools integrated in TEXpert. Over time, management will have to consider and reconcile these forces when making the implementation decision for a knowledge-based system.

Acknowledgment

The authors wish to acknowledge the work and helpful comments of the Texaco AI team: W. S. Dalton, R. L. S. Krog, P. H. G. Thompson, and T. Urwongse.

Cited references

1. *INTELLECT, General Information Manual*, G320-9199, IBM Corporation; available through IBM branch offices.
2. *IBM Expert System Consultation Environment VM and Expert System Development Environment/VM, General Information Manual*, GH20-9597, IBM Corporation; available through IBM branch offices.
3. Shalom M. Weiss and Casimir A. Kulikowski, *A Practical Guide to Building Expert Systems*, Rowman and Allanheld, Publishers, Totowa, NJ (1984).
4. F. Hayes-Roth, D. Waterman, and D. Lenat (editors), *Building Expert Systems*, Addison-Wesley Publishing Co., Reading, MA (1983).
5. *Knowledge Engineering System, User's Manual*, Software Architecture and Engineering, Inc., Arlington, VA (1983).

General references

Avron Barr, Paul R. Cohen, and Edward A. Feigenbaum (editors), *The Handbook of Artificial Intelligence*, William Kaufmann, Inc., Los Altos, CA (1982).

Walter Reitman (editor), *Artificial Intelligence Applications for Business*, Ablex Publishing Co., Norwood, NJ (1984).

IBM Expert System Consultation Environment/VM, User's Guide, SH20-9606, IBM Corporation; available through IBM branch offices.

IBM Expert System Development Environment/VM, User's Guide, SH20-9608, IBM Corporation; available through IBM branch offices.

Interactive System Productivity Facility, Version 2, General Information Manual, IBM Corporation; available through IBM branch offices.

Earl D. Hodil *Texaco Inc., Computer and Information Systems Department, P.O. Box 37327, Houston, Texas 77237.* Since joining Texaco in 1980, Mr. Hodil has served in a number of roles including work in application maintenance and development. In addition to his application systems experience, he has worked on the development of several internally used software tools that aid the development and maintenance process. His current assignment is with the Artificial Intelligence group, where he developed TEXpert. He received a B.A. in economics from Mississippi State University in 1980, and is currently working toward an M.S. in computer science at the University of Houston.

Charles W. Butler *Colorado State University, CIS Department, Fort Collins, Colorado 80523.* Dr. Butler is an associate professor in the Computer Information Systems Department. For the last four years, he has worked on various special projects at Texaco. He previously served industry and government in selected staff positions at the Department of Labor, General Telephone of Florida, and the Tampa Bay Regional Planning Council. Dr. Butler earned his Ph.D. in business administration at Texas A&M University in 1981 after receiving his B.A. and M.S. from the University of South Florida.

Gary L. Richardson *Texaco Inc., Computer and Information Systems Department, P.O. Box 37327, Houston, Texas 77237.* Dr. Richardson is currently Director of Technological Assessment, Research, and Planning for Texaco's Computer and Information Systems Department and has held various senior management positions within the organization since 1979. Prior to this, he served as a professor at Texas A&M University. He has written four computer-related texts and numerous technical articles, and has served as a consultant to the U.S. Air Force, other government agencies, and industry. Dr. Richardson received a B.S. in mechanical engineering from Louisiana Tech University, AFIT training in meteorology at the University of Texas, an M.S. in engineering management at the University of Alaska, and a Ph.D. in business administration–production management at North Texas State University.

Reprint Order No. G321-5268.