Interfaces for knowledge-base builders' control knowledge and application-specific procedures

by P. Hirsch
W. Katke
M. Meier
S. Snyder
R. Stillman

Expert System Environment/VM is an expert system shell—a general-purpose system for constructing and executing expert system applications. An application expert has both factual knowledge about an application and knowledge about how that factual knowledge should be organized and processed. In addition, many applications require application-dependent procedures to access databases or to do specialized processing. An important and novel part of Expert System Environment/VM is the technique used to allow the expert or knowledge-base builder to enter the control knowledge and to interface with application-dependent procedures. This paper discusses these high-level interfaces for the knowledge-base builder.

Introduction

Expert System Environment/VM is an expert system shell—a general-purpose system for constructing and executing expert system applications. It consists of two programs: Expert System Development Environment/VM (ESDE/VM) and Expert System Consultation Environment/VM (ESCE/VM). ESDE/VM provides the functions for the development of a variety of specific expert system applications. ESCE/VM is the consultation portion of the Expert System Environment/VM and allows a user to execute expert system applications on different processors.

Motivation for designing Expert System Environment/VM arose from observing the difficulty with which expert system applications are built—each is often one of a kind and building it requires the skill of a person trained in the area of Artificial Intelligence. There is need for a way to make expert systems easy to build by experts or other people having limited training in data processing, that can be used in a variety of problem areas, and that can still produce systems that run efficiently. Expert System Environment/VM is such an expert system shell. An application expert has both factual knowledge about an application and knowledge about how that factual knowledge should be organized and processed. In addition, many applications require

application-dependent procedures to access databases or to do specialized processing. An important and novel part of ESDE/VM is the technique used to allow the expert or knowledge-base builder to enter the control knowledge and to interface with application-dependent procedures. This paper discusses these high-level interfaces for the knowledge-base builder.

In this paper, we first present an overview of Expert System Environment/VM. After a brief review of some previous efforts at control, we show how ESDE/VM extends this control work through the use of control language primitives and control language strategies. Next we review some of the applications that have used special-purpose interfaces to application-specific data sources. We then show how ESDE/VM improves on these approaches by providing a generalized interface for builder procedures. In the last section we give an example of an expert system that uses the ESDE/VM general-purpose interfaces to produce written documents, such as legal contracts.

Expert system environment overview

Three kinds of people may interact with an expert system shell:

- The client or end user of the application.
- The expert.
- The knowledge engineer, who helps the expert to encode his knowledge.

ESDE/VM is designed to be used by knowledge-base builders (experts or knowledge engineers) to build a variety of expert system applications. The experts may be, for example, insurance underwriters, lawyers, bankers, or engineers. These experts may not have had previous programming experience but have a requirement to build applications based on their expertise. ESDE/VM was designed to be easy to use, with a high-level English-like interface, so that experts can easily build their applications.

The technology of expert systems allows the building of the knowledge base of application facts and relationships to be separated from the processing of the knowledge base. Compared with the traditional application development process of writing code for each application, expert systems technology makes applications easier to build and maintain. This is primarily due to the separation of the complex processing of an application from the knowledge and expertise about the application.

ESDE/VM incorporates a set of procedures that can be used to develop a variety of applications—it is an empty system into which the knowledge-base builder inserts his own rules (to define the "knowledge base") and chooses a reasoning method (the "inference engine"). The result is a specific expert system ready for the end user. ESDE/VM has an intelligent editor program that allows a knowledge-base builder to create the knowledge base in the form of English-like rules, parameters, and controls. The editor verifies the consistency of the knowledge by providing to the knowledge-base builder immediate messages if there is a semantic or syntactic error.

ESDE/VM also includes a set of basic inference-engine-processing functions. Examples include backward chaining (obtaining a value for a desired goal by working backward to the given premises) and forward chaining (proceeding from given data to make inferences and draw conclusions or carry out actions). From these basic inferencing functions, the systems control language permits the building of more complex functions.

It is the application of these functions to the rules in the knowledge base that determines the particular characteristics of the expert system. In each system the base set of knowledge-processing functions is the same; what differs is the sequence and focus of their execution.

The ESDE/VM system runs on an IBM System/370 under the VM/CMS operating system. A knowledge-base builder can use the English-like rules and parameters to construct an application more rapidly than with standard programming techniques. A knowledge engineer can use either of two standard inference techniques, backward or forward chaining, and can control the inference process though a control language. In addition, a knowledge engineer can organize or group the knowledge base into hierarchical structures, called Focus Control Blocks (FCBs). Each FCB can have its own inference engine and its own control steps. ESDE/VM also allows for knowledge-base builder procedures to be easily attached to the system to provide information to or use information from other programs, files, or databases.

An ESDE/VM knowledge base contains knowledge about a particular application domain. This domain knowledge is mainly in the form of knowledge-base objects:

- Parameters—application facts and constraints.
- Rules—relationships among parameters.
- Focus Control Blocks—knowledge-base organization and control specifications.
- Groups—collections of rules, parameters, or focus control blocks.
- Screens—client screens for asking questions or displaying results.

Parameters A parameter has a name and various other properties, such as type (number, boolean, alphanumeric string, binary string, hexadecimal string). Parameters can be single-valued or multivalued and may have a constraint placed on their values as part of their definition. For example, the constraint property of the parameter color could be defined as

```
TAKEN FROM ('periwinkle','persimmon','peach')
```

which means that during a consultation color could only have one of these three values.

Rules Rules specify relationships among parameters, and it is the rules that contain the principal knowledge of a knowledge base. A rule contains a premise part (IF. . . .) and an action part (THEN. . . .), which causes an action to be taken when the premise statement is TRUE.

For example, if there were three parameters in a knowledge base

```
sky
wind
prediction
```

then there could be a rule

```
if sky is 'gray' and wind > 30 then prediction is 'rain'
```

which means that if sky is equal to gray and wind is greater than 30, then set the value of prediction equal to rain.

Focus Control Blocks Focus Control Blocks (FCBs) are the primary building blocks for ESDE/VM control. Each control block contains a collection of rules and parameters, and each control block represents a single focus or unit of work to be accomplished during a consultation. The control blocks are related to each other through being in a hierarchy. For example, in a computer fault diagnostic system, FCBs could represent the CPU, disk drives, tape drives, and communication channels. For an infectious disease system, FCBs could represent the patient, cultures, organisms, and therapies.

Each Focus Control Block can be replicated so that, for example, a collection of rules could be replicated for each culture that was taken for a given patient. The number of instances can be controlled either statically in the control block or dynamically through a rule.

Each FCB has a number of "properties," which can be specified to produce a desired control flow for that particular unit of work. The control text is the major property in that specification. An example of control text is

```
ASK (sky, wind);
DISCOVER;
DISPLAY prediction;
```

which would produce the following sequence of events. First, the client would be asked for the values of sky and wind. Next, the forward chainer would be invoked to find other parameter values, and finally, the value found for prediction would be shown to the client.

FCBs can be organized into a parent/child hierarchy. Also, except for the root FCB, multiple copies (instances) of the same FCB are allowed. An example of an FCB hierarchy is illustrated in **Figure 1**.

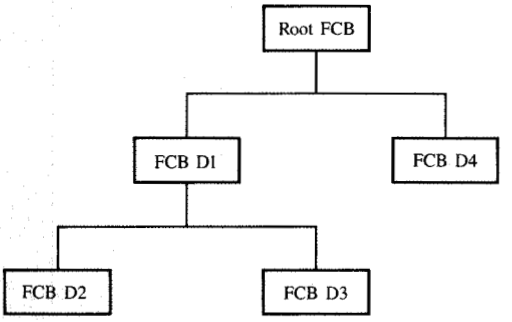

Figure 1

A Focus Control Block hierarchy.

Groups Groups are named collections of similar ESDE/VM objects (parameters, rules, and FCBs). The function of a group is to facilitate reference to a number of similar knowledge-base objects. For example, the names of several parameters can be placed in a group so that they can be referred to by a single name in a rule rather than by a list of individual names.

Screens During a dialog with a client, ESCE/VM can display questions and results of the consultation. The system has built-in default screens. A knowledge-base builder can, in addition, create his own screens of unique design and ask one or more questions on a single screen. The screen is designed with the aid of an interactive screen design editor.

Historical background: control

Some of the early rule-based expert systems, such as MYCIN [1], had much of their control knowledge implicitly specified in their rules. In effect, some of the rules were used as a type of programming language. For these implicit control rules, the premise clause order is very critical. In response to this implicit control problem, Davis [2] proposed the use of meta-rules which can contain strategy information as well as control knowledge. In another approach, the program Amord [3] reverts to a very primitive control mechanism that has almost no built-in preconceptions.

In ONCOCIN [4], control knowledge is separated from domain knowledge by the use of control blocks. The need for screening-type rules is eliminated by the use of contextual information, which defines when a rule can be applied.

In NEOMYCIN [5], control knowledge is represented abstractly, and separately, from domain facts and relationships in basic operating units called tasks. These tasks can form a hierarchical tree structure.

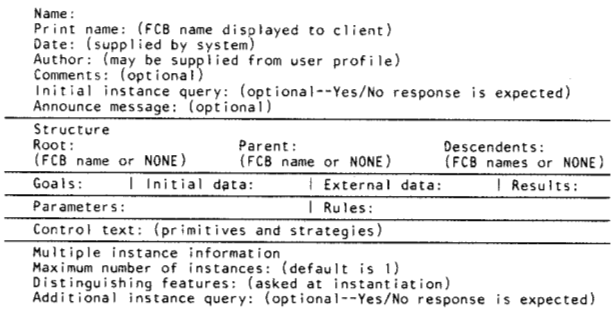

Figure 2

Some Focus Control Block properties.

CRYSALIS [6] is a three-level hierarchical system. Control within this system proceeds from the top level down using strategy rules, task rules, and application domain rules, in that order.

Georgeff [7] proposes a different approach for the control of a system for developing plans or a sequence of actions. His concept is to activate only a subset of the total rule set at any given time.

In a prototype system of Reinstein and Aikins [8], control knowledge is represented as antecedent rules which operate on an agenda of pending tasks. In this system, domain facts and relationships are represented separately in frame hierarchies.

CENTAUR [9] is organized around frame-like objects called prototypes. Knowledge is represented as a combination of these prototypes and rules. Control knowledge is represented separately within each prototype, which gives a context-specific control structure.

ESDE/VM control

ESDE/VM extends the control work done in the previously mentioned systems by providing an explicit, separately manipulable control language in which to express a search strategy and a set of alternative strategies with which to make modifications to the basic control scheme. ESDE/VM Focus Control Blocks are used to organize this control structure.

It is important to distinguish between the operation within ESDE/VM Focus Control Blocks and the operation within contexts, such as in EMYCIN [10]. EMYCIN contexts operate in an implicit, fixed, and prespecified manner. In contrast, the control text in the ESDE/VM FCBs is completely explicit and free-form. This text can be different for each FCB, and it can combine forward chaining, backward chaining, asking for data, displaying results, using external data routines, and invoking other FCBs in any combination, and as many times as desired. This difference will become more apparent when we discuss the ESDE/VM control components in more detail in the following sections.

- *Focus Control Blocks*

FCBs provide an overall control structure for ESDE/VM. Each FCB represents a subtask for a given application. In this respect only, FCBs are analogous to the "contexts" of ONCOCIN [4], the "hypotheses" of NEOMYCIN [5], the "prototypes" of CENTAUR [9], and the "processing states" of the prototype system [8]. Each FCB has a number of "properties," which can be filled in to produce a desired control flow for that particular unit of work. FCBs are typically organized in a hierarchy as defined by the structural properties (root, parent, and descendents) in each FCB. The hierarchy of FCBs is used to resolve references to parameters not associated with the current FCB. Parameters above the current FCB in the FCB hierarchy are visible to that FCB. Each FCB may incorporate groups of parameters defined as goals, initial data, external data, and results, as well as groups of rules. Each FCB also contains a set of control language statements which are used to process the parameters and rules associated with that FCB.

Just as EMYCIN allows multiple instances of its contexts, ESDE/VM also allows multiple instances of an FCB. ESDE/VM extends the EMYCIN work by allowing a user to specify how many instances will be allowed (a positive integer or any number), to automatically stop invocation of instances, and to specify which parameter or group of parameters distinguish one instance from another (distinguishing features). For example, in an infectious disease expert system, there could be any number of culture FCBs, and culture-site could be a distinguishing feature. **Figure 2** shows some of the FCB properties.

- *Control language*

The ESDE/VM control language consists of seven control primitives:

- ACQUIRE—get external data.
- ASK—ask user for data.
- DETERMINE—invoke backward chainer.
- DISCOVER—invoke forward chainer.
- DISPLAY—display data to user.
- ESTABLISH—initiate FCB processing.
- PROCESS—put external data.

All of these primitives, except DISCOVER, can specify groups of objects for their operands and, except for PROCESS, the action specified in each primitive can be modified by strategies. ACQUIRE obtains values from an external data source, ASK interacts with a user to obtain values, DETERMINE uses a backward chaining inference engine plus information about the sources of new values (i.e., from the user, from rules, from defaults, or from an

external data source) to find parameter values, and DISCOVER uses a forward chaining inference engine over a set of rules to find parameter values. Thus the knowledge-base builder can tell the system not only which objects need values, but also what method to use to find those values. The other control primitives show parameter values (DISPLAY), initiate processing of other FCBs (ESTABLISH), and pass control to external data routines (PROCESS).

The control primitives typically operate on parameters, e.g., DETERMINE size, color, weight; but they can also operate on various FCB properties, such as initial data, external data, results, and goals. For example, the statement DETERMINE goals causes the system to determine values for all of the parameters listed in the goals property of the FCB using the backward chaining inference technique. Other examples of control language statements include

ASK initial data,
where initial data is an FCB property containing a list of parameters the values of which will be asked of the user.

DISCOVER
(has no operand, only has strategy options).

DISPLAY results,
where results is an FCB property.

ESTABLISH limit,
where limit is the name of an FCB.

The ACQUIRE and PROCESS control primitives are discussed further in the section on the ESDE/VM external interface.

• *Strategies*

ESDE/VM also contains a set of strategies which can be used to modify statements in the control language:

- DONT ACQUIRE—exclude object(s) from ACQUIRE or DETERMINE.
- DONT ASK—exclude object(s) from ASK or DETERMINE.
- DONT INFER—exclude object(s) from DETERMINE.
- FOCUS ON—consider object(s) first in DETERMINE.
- IGNORE—ignore object(s) during control action.
- ORDER BY—consider object(s) in this order.
- ORDER RULES BY—order rules by this criterion.
- STOP ON—stop on this criterion in DISCOVER.
- USE—use object(s) during control action.

These strategies allow expression of strategical variations on the basic control language statements. There are strategies that specify the order of groups of objects given to a control language statement (FOCUS ON, IGNORE, ORDER BY, and ORDER RULES BY). For example, following is the control statement and associated strategy that will order the group of rules used to find a value for the parameter fault,

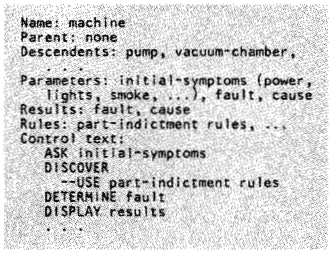

Figure 3

Sample FCB hierarchy for a fault detection problem.

putting those with the least number of unknown parameters in their premise clauses first:

DETERMINE fault
—ORDER RULES BY Least Unknown Premises First.

Other strategies allow a user to alter the specified source of an object during execution of a DETERMINE control language statement (DONT ACQUIRE, DONT ASK, and DONT INFER). As an example, the builder of an infectious disease knowledge base could use the DETERMINE control statement as shown below to find the identity of an organism. If, in addition, one did not want the user of the system to be asked the organism's morphology, one could add a DONT ASK strategy. Finally, if one also felt that the site of the culture was important enough to be found first when there was a choice of parameters to determine, one could add a FOCUS ON strategy:

DETERMINE organism-identity
—DONT ASK morphology
—FOCUS ON culture-site.

The knowledge-base builder can specify a particular group of rules, rather than all rules normally associated with an FCB, to be used with a DISCOVER statement (the USE strategy). In addition, one can define a stopping condition for the DISCOVER statement (STOP ON). For example, if one wanted to forward chain over a set of system-status rules, stopping when the parameter speed became greater than 200, the following control statement and strategies could be used:

DISCOVER
—USE system-status rules
—STOP ON speed > 200.

Sample FCB hierarchy

The fictitious example shown in **Figure 3** for three FCBs in a fault detection problem illustrates some of the control language features of ESDE/VM. The root of the hierarchy in this example is the Focus Control Block machine. Two components of the machine, its pump and its vacuum-chamber, are also represented as Focus Control Blocks. Each FCB has a set of parameters and a set of rules which are used to infer conclusions about the location and cause of the machine fault. Rules and parameters may be listed singly, for example, the temperature parameter and the cable-check rule. ESDE/VM also permits the knowledge-base builder to refer to a group of parameters or rules, for example, initial-symptoms and part-indictment rules. Only a few of the FCB properties are illustrated. A sample of control language is shown for each FCB.

The control interpreter

Operation of the control interpreter is fairly simple. The knowledge-base builder specifies one FCB as the root FCB. This root FCB is invoked (instantiated) at the start of a consultation. Upon entering an FCB, the interpreter begins executing the control language statements in the order specified. Reference to an object not contained in the FCB or any of its parents causes an immediate change in focus to the FCB which contains that object. The interpreter continues invoking FCBs and executing their control language statements until it has completed all work generated from the initial FCB.

Historical background: external data

The simplest example of a special-purpose interface is to pass data by file transfer. The DENDRAL [11] and PUFF [12] programs use this technique. The ONCOCIN [4] and ACE [13] programs are examples of expert systems which have special-purpose interfaces to interact with external database information.

Some expert system applications require the use of sensor-based data, and some of the newer systems are also moving into the real-time domain, where large quantities of rapidly changing data must be efficiently processed. The Serum Protein Diagnostic Program [14] obtains its sensor-based data directly from the scanning densitometer itself. VM [15] was specifically designed to interpret sensor-based data obtained in an intensive care unit. PDS [16] is used for the on-line diagnosis of a chemical process operation on a real-time basis. NDS [17] is an expert system for fault isolation in a nationwide communications network. YES/MVS [18] is a continuous real-time expert system used to assist computer operators in controlling an operating system.

An increasingly important expert system application area involves the external interaction of a consultation or tutoring system with a computer simulation model or a collection of application programs. SOPHIE [19] is used to teach problem-solving skills in a simulated electronics laboratory environment. REACTOR [20] is intended to assist plant operators in handling nuclear reactor accidents. ELAS [21] integrates a production rule advice system directly with pre-existing AMOCO programs for well-log analysis and display. STEAMER [22] is an intelligent computer-based training system which serves as an assistant in naval propulsion engineering instruction.

The processing and interpretation of signal data is another area of current interest in expert system research. HASP/SIAP [23] is one of these programs; it is concerned with the signal-to-symbol transformation problem. CRYSALIS [6] automates the electron density map signal interpretation process. LITHO [24] is a program for interpretation of oil-well log data. The DIPMETER ADVISOR [25] is a commercial expert system which is also a well-log interpretation program.

ESDE/VM external interface

All of the expert system programs mentioned in the previous section used special-purpose interfaces for sourcing external data. In contrast, ESDE/VM does not have this restriction since it provides three generalized procedural escape options: the external data sourcing sequence specification, and both the ACQUIRE and the PROCESS control language primitive commands. ESDE/VM also provides some general-service routines for use in the knowledge-base builder's application-specific procedures.

- *External data routines*

An ESDE/VM parameter can obtain its value in several ways. This sourcing sequence is specified at the time the knowledge base is developed by the knowledge-base builder. Any or all of the sources

rule consequent
user input
apply default
external data

can be specified in any order. Thus, the first method of getting data from an external source is provided by the sourcing sequence property of a parameter. When the ESDE/VM backward chaining inference process encounters this external data option during a consultation run, it tries to obtain the parameter value by invoking a specified procedure. This procedural escape entry point is specified in the procedure name property of the given parameter. Any additional information needed by the external data routine, such as file name and file type, can be specified in the procedure arguments property of that same parameter.

Two of the control language primitives, ACQUIRE and PROCESS, operate on external data and provide the second and third external interfacing options for ESDE/VM.

The ACQUIRE control language primitive command is used to acquire ESDE/VM parameter values from an external data source. The form of this command is

ACQUIRE External-Data USING Procedure-Name

where Procedure-Name specifies the procedural escape entry point. External-Data is an FCB component containing either a parameter name or a list of parameter names whose values are to be obtained from the specified external data source. Again, additional information can be specified in the procedure arguments property of a given parameter.

The PROCESS control language primitive command is used to pass control to an external routine. This option provides a way for an external routine to access ESDE/VM results during or after a consultation session. The form of this command is

PROCESS ESDE/VM-Results USING Procedure-Name

As before, Procedure-Name specifies the procedural escape entry point. ESDE/VM-Results represents either a parameter or a list of parameters whose values are to be made available to the external routine.

• *External data services*

ESDE/VM provides a set of general-service routines which can be used in the external data procedures to obtain and assign ESDE/VM parameter values and to perform other utility functions. The functions of these routines are as follows:

Utility services

- Get the name of a parameter that is the subject of either the backward chaining facility or the ACQUIRE command.
- Get the number of arguments being passed to the routine.
- Get the name of a parameter that is an argument.
- Get the argument data type (string, number, etc.).
- Get the number of values of an argument.
- Put a message into the trace file.

Services that obtain an argument value

- Get an alphanumeric string value.
- Get a number value.
- Get the certainty value of a boolean parameter.
- Get a hexadecimal string value.
- Get a binary string value.

Services that assign a valid value

- Set an alphanumeric string parameter value.
- Set a number parameter value.
- Set a boolean parameter certainty.
- Set a hexadecimal string parameter value.
- Set a binary string parameter value.

Individual get/set commands were used, rather than a simple generic get/set call, because of the strong data typing characteristic of Pascal and the desire to avoid the use of complex data structures. Also, it should be noted that this general-service routine approach means that users do not have to know ESDE/VM internal data structures, pointers, etc., in order to write an external data program.

• *Installing knowledge-base builder procedures*

Knowledge-base builder application-specific procedures can be written in any language that can be called from Pascal/VS. Such languages include Pascal, PL/I, FORTRAN, COBOL, and assembler. ESDE/VM has an external data segment which contains a series of skeleton Pascal procedures (i.e., procedures that contain no Pascal code) into which user-written external procedures can be placed.

ESDE/VM also provides two EXECs and a macro to help integrate the knowledge-base builder procedures with the knowledge base. One EXEC is used to compile the external data segment, and the other one is used for linking the segment to ESDE/VM. The macro contains the library routines needed when compiling the segment.

These user-written routines can call other subroutines available to the user which are not in ESDE/VM. These routines can also exit to user EXECs which contain simple lists of commands that could, for example, be used to make database inquiries.

• *Knowledge-base builder procedures: an example*

A knowledge base has been developed to produce legal contracts for joint study activities with IBM customers, such as universities and research organizations. Each such legal contract is based both on some prepared sections of text that are of a general nature and on some text based on the particular circumstances for that customer contract.

To achieve this text integration, it is necessary to use procedures that are specifically written for this contract application to provide the output of the expert system to IBM's word processor, Documentation Composition Facility (DCF).

Typical rules for such a legal contract system are as follows:

```
IF output_owner is 'IBM'
  THEN there is strong evidence that copyright is 'IBM
  owns'

IF joint_effort is 'Jointly defined only'
  and copyright is 'IBM owns'
  and delivered_software is 'Marketable software'
  and underlying_assumptions is true
  and patents is 'Customer owns'
  and patents is 'IBM has an unrestricted free license'
    THEN there is strong evidence that contract_type is
    'Jointly Defined, Software produced, IBM owns'
```

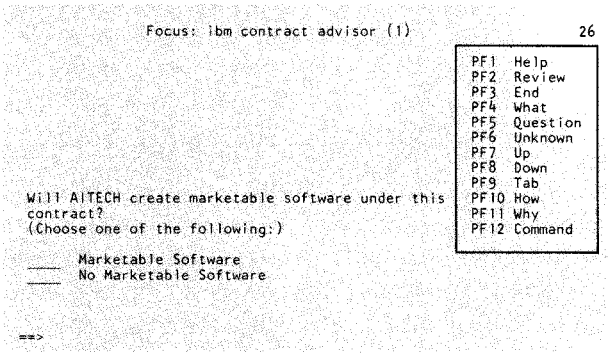

Figure 4

Example of ESCE/VM client screen.

The English-like syntax of these rules allows a contract relations person to build and maintain the application knowledge base.

A typical end-user screen is shown in **Figure 4**.

The end user would then answer this and the subsequent consultation questions to produce the final contract.

The dialog and system processing are driven by the ESDE/VM Focus Control Blocks. These FCBs were developed by the knowledge-base builder to process the rules and parameters specified in the knowledge base. The statements in this particular control text were as follows:

```
ASK initial data;

DETERMINE goals;

DISPLAY (contract_type,initial data,
        royalties,third_party,customer_confidential,
        pre_existing_work,sponsor,
        academic_institution,copyright,patents,
        joint_effort,delivered_software,
        subsidiaries,goals);

PROCESS (fname,ftype,script_id,output2) USING writefile
```

The control text above provides the mechanism and sequence of events for ESCE/VM to process the knowledge that it has in its knowledge base as well as the knowledge that it acquires from the client. The first statement, ASK initial data, asks the user about parameters such as the name of the customer, the customer address, or the starting date of the contract. The second statement, DETERMINE goals, indicates to ESCE/VM that the second order of business is to use the backward chaining inference engine to obtain the necessary goals, such as the type of contract or who should own the patent rights. The third statement in the control text is to DISPLAY (contract_type . . .); which will display to the user the results of the consultation for review. The last statement of the control text is where the external procedure comes into play. The control text is PROCESS (argument list) USING writefile. In this statement, the results of the consultation contained in the parameter output2 are processed and written to a file using a special external procedure called writefile. The program writefile was written in PASCAL specifically for this application. This is the mechanism that Expert System Environment/VM employs to tie together the results of the dialog with the word processing system.

The resulting file has the following form:

```
.se compamt =   '$500,000'
```

September 1, 1985

Professor Calculus
Director of Special Studies
AITECH
1151 Vista Drive
Menlo Park, CA 94303

Dear Professor Calculus:

This letter sets forth the Agreement between AITECH and International Business Machines Corporation ("IBM") relating to a Joint Effort to select weather models using expert systems. It is the desire of IBM and AITECH to gain further knowledge concerning the subject matter of the Joint Effort. The knowledge and materials IBM gains or receives as a result of this Joint Effort may be used by IBM in the design, manufacture, use, rental, lease, license, sale, and application of information processing and related products and services. The parties agree as follows:

1. SCOPE OF WORK

The scope of work, including the Joint Effort objectives, tasks to be performed, responsibilities of each party, Deliverables, task/Deliverables schedule, and personnel requirements, is set forth in Appendix 1 hereto which forms a part of this Agreement.

2. COMPENSATION

2.1 IBM shall pay AITECH up to $500,000 in compensation for the performance of the tasks set forth in Appendix 1 of this Agreement. Such payment shall include the following items: travel, salaries, computer time. This payment, based on the expenses outlined in Appendix 2, shall be made as follows:

Amount	Payment Date
$200,000	September 1, 1985
$125,000	May 1, 1986
$175,000	August 31, 1986

2.2 IBM shall reimburse AITECH for all reasonable travel and living expenses authorized in advance by IBM and incurred in connection with this Agreement. It is understood that reimbursements for travel and living expenses shall be in amounts which are consistent with those received by IBM employees performing comparable services on behalf of IBM.

2.3 AITECH will submit invoices to Bette Jones, 1530 Page Mill Road, Palo Alto, CA 94304, according to the schedule in Section 2.1 above for the services furnished, and monthly for the travel and living expenses incurred hereunder during the preceding month, accompanied by vouchers evidencing such expenses. IBM shall make payment to AITECH within thirty (30) days after receipt of its invoice.

3. INFORMATION TRANSFERS

3.1 All information supplied by either party shall be

. . .

Figure 5

Sample contract.

```
.se comdate3 =  'August 31, 1986'
.se compamt3 = '$175,000'
.se compitem =  'travel, salaries, computer time'
.se comdatel =  'September 1, 1985'
.se compamtl =  '$200,000'
.se comdate2 =  'May 1, 1986'
.se compamt2 = '$125,000'
.cs 2 include
.cs 5 ignore
    ⋮
```

The statements are in the form required by the word processing system, DCF, and indicate that the variable "compamt" (compensation amount) should be set to $500,000 and that Section 2 should be included, but Section 5 should not be included.

The word processing system uses these statements along with the standard text to meld together a contract, a portion of which is shown in **Figure 5**. Notice that the type of contract has been correctly selected, Section 2.2 on travel has been included, and variable names such as compensation amount have been substituted appropriately in the text.

Although this is a fictitious contract with a fictitious customer, AITECH, the language is similar to the actual language contained in many IBM joint study contracts. By employing this type of system, an end user can provide a finished contract to a customer in a shorter period of time with the correct terms and conditions. The ability of these expert systems to integrate easily with existing computer systems adds additional power to a useful technology.

Other Expert System Environment/VM applications have made use of general-purpose graphics procedures, large real-time data bases, and system hardware modeling programs.

Summary

ESDE/VM's explicit representation of control knowledge in Focus Control Blocks allows the knowledge-base builder to separate an expert system application into discrete units of work, each with its own set of objects and control language statements. In addition, strategy options allow the specification of variations on the control language statements. This explicit representation makes the expert system design more transparent and facilitates easy modification of the control strategy.

ESDE/VM's general technique for interfacing with external procedures provides an effective way to use already available computer data and to send ESDE/VM results to other external routines. For this purpose, ESDE/VM provides three options: the parameter sourcing sequence specification, the ACQUIRE control language primitive command, and the PROCESS control language primitive command. ESDE/VM also provides a set of general-service routines which can be used by the external procedures to access ESDE/VM data and to perform various utility functions.

Acknowledgments

Jan Aikins and Harry Reinstein helped to develop ESDE/VM's control architecture, and Mary Degener provided the contract example.

References

1. E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York, 1976.
2. R. Davis, "Meta-Rules: Reasoning About Control," *Artif. Intell.* **15,** 179–222 (1980).
3. J. de Kleer, J. Doyle, G. L. Steele, Jr., and G. J. Sussman, "AMORD: Explicit Control of Reasoning," *SIGPLAN/SIGART Special Issue*, pp. 116–125 (August 1977).
4. E. H. Shortliffe, A. C. Scott, M. B. Bischoff, A. B. Campbell, W. van Melle, and C. D. Jacobs, "ONCOCIN: An Expert System for Oncology Protocol Management," *Proc. IJCAI-81*, pp. 876–881 (1981).
5. W. J. Clancey, "The Advantages of Abstract Control Knowledge in Expert System Design," *Proc. AAAI-83*, pp. 74–78 (1983).
6. A. Terry, "The CRYSALIS Project: Hierarchical Control of Production Systems," *Heuristic Programming Project Report No. HPP-83-19*, Computer Science Department, Stanford University, CA, May 1983.
7. M. P. Georgeff, "Procedural Control in Production Systems," *Artif. Intell.* **18,** 175–201 (1982).
8. H. C. Reinstein and J. S. Aikins, "Application Design: Issues in Expert System Architecture," *Proc. IJCAI-81*, pp. 888–892 (1981).
9. J. S. Aikins, "Prototypical Knowledge for Expert Systems," *Artif. Intell.* **20,** 163–210 (1983).
10. W. van Melle, "A Domain-Independent System That Aids in Constructing Knowledge-Based Consultation Programs," *Heuristic Programming Project Report No. HPP-80-22*, Computer Science Department, Stanford University, CA, June 1980.
11. R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg, *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*, William Kaufmann, Inc., Los Altos, CA, 1980.
12. J. S. Aikins, J. C. Kunz, E. H. Shortliffe, and R. J. Fallat, "PUFF: An Expert System for Interpretation of Pulmonary Function Data," *Heuristic Programming Project Report No. HPP-82-13*, Computer Science Department, Stanford University, CA, September 1982.
13. G. T. Vesonder, S. J. Stolfo, J. E. Zielinski, F. D. Miller, and D. H. Copp, "ACE: An Expert System for Telephone Cable Maintenance," *Proc. IJCAI-83*, pp. 116–121 (1983).
14. S. M. Weiss, C. A. Kulikowski, and R. S. Galen, "Developing Microprocessor Based Expert Models for Instrument Interpretation," *Proc. IJCAI-81*, pp. 853–855 (1981).
15. E. H. Shortliffe and L. M. Fagan, "Expert Systems Research: Modeling the Medical Decision Making Pocess," *Heuristic Programming Project Report No. HPP-82-3*, Computer Science Department, Stanford University, CA, March 1982.
16. M. S. Fox, S. Lowenfeld, and P. Kleinosky, "Techniques for Sensor-Based Diagnosis," *Proc. IJCAI-83*, pp. 158–163 (1983).
17. T. L. Williams, P. J. Orgren, and C. L. Smith, "Diagnosis of Multiple Faults in a Nationwide Communications Network," *Proc. IJCAI-83*, pp. 179–181 (1983).
18. R. L. Ennis, J. H. Griesmer, S. J. Hong, M. Karnaugh, J. K. Kastner, D. A. Klein, K. R. Milliken, M. I. Schor, and H. M. Van Woerkom, "A Continuous Real-Time Expert System for Computer Operations," *IBM J. Res. Develop.* **30,** 14–28 (1986, this issue).
19. J. S. Brown, R. R. Burton, and J. de Kleer, "Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III," *Intelligent Tutoring Systems*, D. Sleeman and J. S. Brown, Eds., Academic Press, London, 1982, pp. 227–282.

20. W. P. Nelson, "REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents," *Proc. AAAI-82*, pp. 296–301 (1982).
21. S. Weiss, C. Kulikowski, C. Apte, M. Uschoid, J. Patchett, R. Brigham, and B. Spitzer, "Building Expert Systems for Controlling Complex Programs," *Proc. AAAI-82*, pp. 322–326 (1982).
22. J. D. Hollan, E. L. Hutchins, and L. Weitzman, "STEAMER: An Interactive Inspectable Simulation-Based Training System," *The AI Magazine* **5**, No. 2, 15–27 (Summer 1984).
23. H. P. Nii, E. A. Feigenbaum, J. J. Anton, and A. J. Rockmore, "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *The AI Magazine* **3**, No. 2, 23–35 (Spring 1982).
24. A. Bonnet and C. Dahan, "Oil-Well Data Interpretation Using Expert System and Pattern Recognition Technique," *Proc. IJCAI-83*, pp. 185–189 (1983).
25. R. G. Smith, "On the Development of Commercial Expert Systems," *The AI Magazine* **5**, 61–73 (Fall 1984).

Received April 23, 1985; revised September 3, 1985

Peter M. Hirsch *IBM Scientific Center, P.O. Box 10500, Palo Alto, California 94304.* Dr. Hirsch received his B.Sc. in applied mathematics and engineering physics in 1961, his M.Sc. in mathematics in 1966, and his Ph.D. in numerical analysis in 1966, all from the University of Wisconsin, Madison. Prior to joining IBM in 1966, he was an instructor in the Mathematics Department at Pennsylvania State University. He joined IBM at the Houston, Texas, Scientific Center, working in the areas of power application and computer generated holograms. Since joining the Palo Alto Scientific Center in 1974, he has worked on expert systems, power systems analysis, and energy. He received an IBM Division Headquarters Excellence Award in 1982 for the development of FORTRAN Utilities for VM, an IBM Division Headquarters Excellence Award in 1979, an IBM Outstanding Invention Award in 1969, an IBM First-Level Invention Achievement Award in 1970, and an IBM Second-Level Invention Achievement Award in 1971. Dr. Hirsch is a member of the American Association for Artificial Intelligence and the Association for Computing Machinery.

William Katke *IBM Scientific Center, P.O. Box 10500, Palo Alto, California 94304.* Mr. Katke has been involved in artificial intelligence since 1967. While at the University of Wisconsin, he worked on Autoling, a program that learns a syntax for natural language. He also conceived and developed a program generation system using semantic networks. Since 1974, he has worked for the World Bank, the Computer Science Corporation, Advanced Computer Techniques, and the Planning Research Corporation. His experience includes the design of a back-end database system, the design and development of a distributed electronic mail system for IBM PCs, and the development of MsSpeller, a spelling correction program for the Apple II. He joined IBM in 1984 to participate in the development of Expert System Environment/VM, which has been announced as a program offering. Mr. Katke received a Director's Award for his work on Expert System Environment/VM.

Michael Meier *IBM Scientific Center, P.O. Box 10500, Palo Alto, California 94304.* Mr. Meier is a staff member in the expert systems group at the Palo Alto Scientific Center, where he is involved in the development of the Expert System Environment/VM. Prior to this assignment, he worked in the service research knowledge-based systems group on a prototype of the expert system environment called Prototype Inference System (PRISM). Mr. Meier joined IBM in 1978 as an MVS program support representative in the Southfield, Michigan, Field Engineering branch office. Before joining IBM, Mr. Meier was with Kimberly-Clark Corporation and Federal-Mogul Corporation as a system programmer working primarily on IMS and MVS. He has a B.S. in mathematics/computer science from the Lawrence Institute of Technology, Southfield, Michigan.

Steven Snyder *IBM Scientific Center, P.O. Box 10500, Palo Alto, California 94304.* At present Mr. Snyder is working in the area of expert systems. He defines requirements for, implements, and maintains an experimental expert systems shell (PRISM). From 1980 to 1984, he was with Field Engineering Service Research in Palo Alto investigating expert systems techniques, benefits, potential application areas, and requirements. From 1965 to 1980, he held a variety of positions with the Systems Manufacturing Division, the Field Engineering Division, Systems Development Division, System Communications Division, Data Processing Division, and the Field Engineering Systems Center. Prior to joining IBM, Mr. Snyder was with the U.S. Army and Allied Chemical Corporation, Buffalo, New York. He attended the State University Agricultural and Technical College, Alfred, New York, in 1959, the State University of New York, Buffalo, from 1960 to 1962, and Arizona State University, Tempe, from 1968 to 1969. In 1985 he received an IBM Outstanding Technical Achievement Award for his work on the Expert System Environment/VM program offering. Mr. Snyder is a member of the American Association for Artificial Intelligence.

Richard E. Stillman *IBM Scientific Center, P.O. Box 10500, Palo Alto, California 94304.* Dr. Stillman joined the IBM Research Division in 1958 in a process control group. He is currently helping to apply expert systems technology to IBM manufacturing productivity improvement. Although he is located at the Palo Alto Scientific Center, he reports to Manufacturing Research in Yorktown Heights, New York. His past work has been in the areas of expert systems, process simulation and optimization, and numerical solution of differential equations. Dr. Stillman received his B.S. from the University of Kansas, Lawrence, in 1951, his M.S. from the University of Kansas in 1956, and his Ph.D. from the Pennsylvania State University, University Park, in 1969, all in chemical engineering. He is a member of the American Association for Artificial Intelligence, the American Institute of Chemical Engineers, Phi Lambda Upsilon, Sigma Tau, Sigma Xi, and Tau Beta Pi.