CLIPS Interface Development Tools and Their Application

Bernard A. Engel [1], Chris C. Rewerts, Raghavan Srinivasan, Joseph B. Rogers, and Don D. Jones

Abstract

A package of C-based PC user interface development functions has been developed and integrated into CLIPS. The primary function is **ask** which provides a means to ask the user questions via multiple choice menus or the keyboard and then return the user response to CLIPS. A parameter-like structure supplies information for the interface. Another function, **show**, provides a means to paginate and display text. A third function, **title**, formats and displays title screens. A similar set of C-based functions that are more general and thus will run on UNIX and other machines have also been developed. Seven expert system applications were transformed from commercial development environments into CLIPS and utilize **ask**, **show**, and **title**. Development of numerous new expert system applications using CLIPS and these interface functions has started. These functions greatly reduce the time required to build interfaces for CLIPS applications.

Introduction

The Agricultural Engineering Department at Purdue University has been developing agricultural expert systems (ES) applications since 1984. Numerous applications have been developed since that time including GMA (Grain Marketing Advisor), DSS (Dam Site Selector), DBL-CROP (double crop soybean management ES), and MELON (muskmelon disease diagnosis ES). The use of these and other applications has been limited because of runtime licensing fees, inability to run on machines other than those running DOS, and difficulties with integrating applications with other software. In addition, the academic community has not been interested in the paperwork associated with the licensing arrangements of most ES development tools. Many of the commercial tools require development and delivery on a single type of machine such as one that runs only DOS. In Indiana, the Cooperative Extension Service (a large potential user for most agricultural ES) have UNIX machines. Therefore, many of the agricultural ES that would be of interest to people in these offices will not run on their machines. The ES that are now being developed often require integration with other computer tools. Most commercial ES development tools do not provide adequate facilities for integration. As a result of these problems, CLIPS was examined as a potential development and delivery tool for agricultural expert systems applications.

One disadvantage of CLIPS, for our purposes, was the lack of a cost-free end-user interface for use with PC compatible machines. The interface available for the PC version of CLIPS requires the purchase of a screen-handling command library from a third-party vendor. End-users of ES produced with the interface provided would have to pay a fee for its use. Also, it was more a development interface than an end-user interface. To avoid such complications, we have built a set of interface functions and integrated them with CLIPS.

---

1 The authors are: B.A. Engel, Assistant Professor; C.C. Rewerts, Research Assistant; R. Srinivasan, Research Assistant; J.B. Rogers, AI Systems Programmer; and D.D. Jones, Professor. Agricultural Engineering Department, Purdue University, West Lafayette, IN 47907

User Interfaces

For a computer program to function, it must interface with an outside manipulator or controller. Some programs are controlled by other programs. An interface needed in such situations can typically be described outright, in well-defined terms. The interchanges will be predictable, because machines are involved. In a computer program developed for use by humans, the interface becomes a much different issue. The operation of the interface has a direct bearing on how well one can make use of the program. The user must be able to "run" the program while providing any needed inputs and making any requests for modification of runtime functions. There are many common programs with simple operations that function automatically when invoked, such as using *ls* or *dir* to list files in a DOS directory. However, the programs we refer to in the context of this paper are application programs requiring more user-machine interaction during program operation, such as an ES or simulation. In such cases, "the interface *is* the system for most users" [1].

The Need for an End-User Interface Package

In developing numerous application programs for distribution to a large audience of users with a wide range of computer backgrounds, we needed an interface package that could be incorporated into separate application programs. Of course, we were not the first to discover this need. In working with the design of several software projects, Faneud and Kirk [2] noted the following complaints:

a. Interface development was consuming a great part of the efforts of ES developers and represented a significant portion of the resulting code - as much as 60%.

b. ES developers were usually inexperienced at interface design, and generally had no interest in becoming experts in low level graphics or other interface tools.

c. There was no consistency of interfaces across applications.

d. It was difficult to provide multiple interfaces across applications.

Some of the benefits we hoped to gain by the development of a user interface that could be used by numerous applications included:

a. Users can employ a small number of computer concepts and syntactical rules, therefore they can concentrate on the task.

b. Program designers find it convenient to reduce the number of situations in which the user can make errors.

c. Different applications using the same interface package will have the same "look and feel" to the user. Thus, once a user becomes acquainted with an interface through the use of one application, the use of subsequent programs with the same type of interface may be made simpler.

Implementation and Development Background

This paper documents a simple user interface and its integration into CLIPS. Although all examples and most of the discussion of the user interface will revolve around its implementation within CLIPS, applications are not limited to ES. Most computer applications designed for a general end-user audience require an interface of one sort or another. With an ES, the operation typically starts with a question and answer session between the user and the program, much like a human expert would use to ascertain the definition of the problem to be addressed. A good interface package would allow the developer of the ES a straight-forward means to define how the ES should go about the task of this interchange. We will describe how the interface and CLIPS communicate, how the interface functions are used from within a CLIPS ES program, and how the interface presents its information and queries to the end-user.

Appearance of Ask to the End-User

Our user interface is designed to use the graphics capabilities of the PC on which it is running, including high resolution graphics, if available. The interface presents itself in color, if available. When graphic capabilities are available on a given machine, provisions have been made to use graphic screens instead of, or in conjunction with, the textual screens used for "help", "why", or the question prompt.

The layout of the ask interface screen consists of three areas:

1. An information box,
2. The question prompt, and
3. An area for the input of the user's answer.

The information box (Figure 1), located at the top of the screen, informs the user of the "help", "why", and "abort" keys (F1, F2, and ESC, respectively). Based on the type of question, the information box tells the user what type of input is expected. In Figure 1, the input expected is a single selection from the three alternatives in the menu box. The second line in the information box gives brief instructions for selecting an answer.

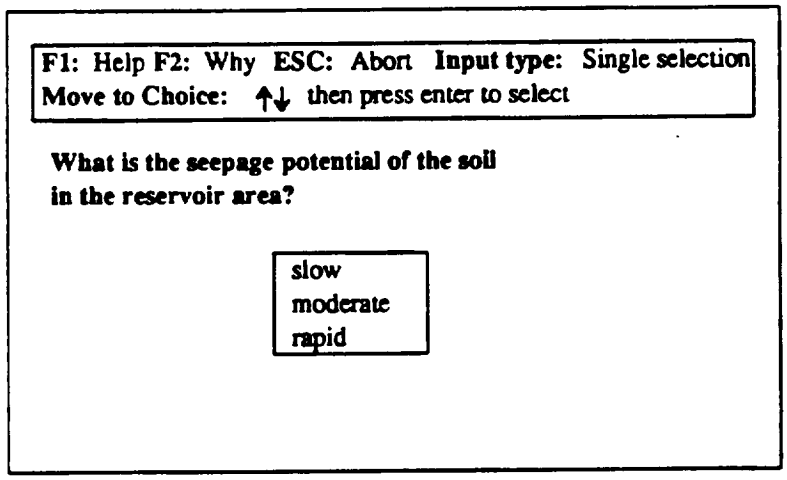

Figure 1. Layout of an ask question screen.

If the user presses the F1 or F2 key, the ask program switches to a display screen to print the information provided by the knowledge engineer for the particular question property structure. "Help" or "why" information may be a graphic image, text, both, or neither. If the ESC key is pressed, the intent of the user to abort the program is confirmed with a dialogue box. If confirmed, both the operation of the interface *and* the operation of CLIPS is aborted.

The text of the question is printed in the question prompt area. Ask provides formatting to fit the text neatly on the screen. Below the question prompt is the user input area. It will be a menu box if the question type requires selection from a list of alternatives. Two types of menus are available in ask, one allows the user to select a single selection as an answer, and the other allows multiple selections. To select a single answer from the menu box, the user moves to a choice with the mouse or "up-down" arrow keys to highlight a choice. When the user presses the enter key, the highlighted selection is returned to CLIPS. To select multiple answers from a menu, the user may "mark" a highlighted selection with the "right" arrow key. All selections either marked or highlighted when the "enter" key is hit will be returned to CLIPS.

Menu selection is appropriate only when specific answers are expected. To obtain more open-ended responses, ask can prompt for input of text or a number. In this case, a simple prompt for the information is printed in the user input area. Ask can be given a range for numerical entry, and will constrain the user's entry as needed.

Errors

All interface functions are equipped with abilities to detect and report errors. An error is generated when the information being passed to a function is inconsistent with the expected format. (These errors will generally pertain to problems most likely to arise during development of an ES). The action taken by the functions in case of an error is to abort all processes and print a diagnostic statement.

Using Ask in an Expert System

Ask is invoked with a frame-like parameter structure that passes it the information it needs to operate. One of the first things that must be determined is the type of question screen it is to construct. As mentioned above, the four types of question screens ask generates are:

1. Multiple choice/single answer,
2. Multiple choice/multiple answer,
3. User input of text,
4. User input of numeric data.

For each question screen, a data structure for the ask function must be written. The data structures are stored in CLIPS as *facts*. The data structure will tell the ask function how to formulate the question, what kinds of extra information to provide to the user, how to retrieve the user's answer, and how to return the resulting answer to CLIPS.

Creating Instructions for Ask

We will refer to the above-mentioned data structures as "*question property structures*". CLIPS facts are stored as "*fields*", where each field is a word, number or "*string*" (a group of words or numbers contained in double quotes). When ask is given a question property structure from which to build a question prompt, it examines the fields one by one, looking for the information it needs.

There are two types of fields expected:

1. labels: Labels are key words used to identify the information that may follow in the next field(s). The ask function expects exactly thirteen labels.
2. values: Values are the actual information ask will use to construct and ask the given question. The ask function requires some labels to be followed by values, some label's values may be a certain type, and some may be ignored by the ask function (because they may apply elsewhere in the ES or are reserved for a future use).

As ask reads through the labels and values of a question property structure, it deduces what type of question it is to ask, based on the values. Table 1 is the list of labels and values, and how they are used by the ask function.

Constructing Question Property Structures

To use the ask function in a CLIPS program, question property structures must be entered as *facts*. Examples 1 through 4 demonstrate question structures in their format as *facts*. Each example will produce a different type of ask question screen.

Example 1.

```
(site-name
prompt
"What is the name of the site you wish to evaluate?"
expect
help
why
value
value-type
default
range
certainty-range
unknown
gprompt ghelp gwhy
)
```

The above example illustrates a question property structure. The first field, (in this case, *site-name*), can be any *word*, which is to say, any combination of legal characters, with no spaces. The purpose of this word is to label the question property structure, so that a rule could be constructed to look for a fact starting with the given word, which it could match and fire (this is explained further in the discussion of the example rule, below).

This example demonstrates the simplest type of question property structure, because it uses the least amount of information allowed: the labels, and a string value (the question) for the prompt. Values for all parameters except prompt are optional. Since no values are given for the *expect* label, ask deduced the question was to be answered by user input of text. To answer the question from the ask-generated interface, the user types in an answer, and presses the "enter" key.

Example 2.

```
(seepage-rate
prompt
"What is the seepage potential of the soil in the reservoir area?"
expect
slow moderate rapid
help
"A soil survey of the proposed reservoir site should provide
information concerning the seepage rate of soil at the site.
However, if the soil survey does not provide this information
answer the question as not being certain and additional
questions will be asked to evaluate the seepage rate."
why
value
value-type SINGLEVALUED
default
range
certainty-range
unknown
gprompt ghelp gwhy
)
```

There are three primary differences between this question property structure (Example 2) and the last example:

1. Three values are listed after the expect prompt, "slow", "moderate", and "rapid". When **ask** reads these values, it will set up a menu-type question, with the values to choose from.
2. Following the *help* label, is a value in the form of a string, "A soil survey of the proposed reservoir...". This is to be the help message displayed when the **F1** key is pressed when the question is asked. To include *why* information in a question property structure, use the same method as for *help*. To view *why* information while answering a question, the user presses the **F2** key.
3. The third difference is the value *SINGLEVALUED* following the value-type label. This tells the **ask** function to allow the user to select only one of the *expect* choices.

Example 2's question property structure produces a question with a menu-type answer selection (Figure 1). To answer the question, the user points to a selection with a mouse or uses the up/down arrow keys to highlight a choice, then the enter key to select the highlighted choice. The selection is returned to a **CLIPS** rule for processing.

Example 3 is a question property structure that will trigger **ask** to create a question that asks the user to input a number. This was done by setting the *value-type* value to *NUMERIC*. Since the expected answers on many questions asking for numerical input will fall within some range, it is logical to set the *range* values. In this case, if the user tries to enter a number outside the range of 1 to 10000, **ask** will inform the user of the range imposed and prompt the user to try again.

Example 3 offers two types of help to the user, text and graphic. The text can be seen following the *help* label. The name of the graphic image file, "area.hlp" appears following the label *ghelp*. "Area.hlp" is the name of the graphic image file to be shown to the end-user if help is requested.

Example 3.

```
(surface-area
prompt
"What is the surface area of the reservoir, in acres,
if the water in the reservoir is at its normal depth?"
expect
help
"To determine the surface area of the proposed reservoir
at its normal depth, a survey of the area or a blown-up
USGS map of the reservoir site is needed. A planimeter
should be used to determine the area from the survey or map."
why
value
value-type NUMERIC
default
range 1 10000
certainty-range
unknown
gprompt
ghelp area.hlp
gwhy
)
```

Example 4.

```
(water-use
prompt
"What is the intended use of the water that will be impounded
in the reservoir?"
expect
water-supply recreation flood-control
help
"More than one of the expected values can be selected."
why
value
value-type MULTIVALUED
default
range
certainty-range
unknown YES
gprompt
ghelp
gwhy
)
```

Example 4 causes the ask function to generate a question menu that allows the user to choose more than one option, because the value for *value-type* is set to *MULTIVALUED*. Another feature of the question is that it offers the option "unknown" in the menu, as well as the listed *expect* options "water-supply", "recreation", and "flood-control". This is because the value "YES" appears after the label "unknown".

To answer this question with multiple answers, the user selects choices by highlighting a choice, and pressing the right arrow key. This "marks" the highlighted selection. (Inversely, if the left arrow key is pressed, a highlighted choice is "un-marked"). Other choices can be highlighted and marked. When enter is pressed, all choices that are highlighted or marked are returned to CLIPS as the answer.

Example 5. An Example Program Using Ask

```
(deffacts menus
(site-name
 ...rest of question property structure...)
(seepage-rate
 ...rest of question property structure...)
(surface-area
 ...rest of question property structure...)
(water-use
 ...rest of question property structure...))

(defrule interrogator-rule
        ?d <-(?question-name prompt $?question-prop-strct)
        =>
        (bind ?result (ask $?question-prop-strct))
        (assert (?question-name ?result))
        (retract ?d))
```

Points of Interest

Our example program consists of only four *facts* (Examples 1 through 4) and one *rule*. For this reason it is practical to put all the necessary information in one knowledge base file. The CLIPS command, *deffacts*, defines information to be loaded as facts. (There are other ways to load or enter facts into the CLIPS knowledge base, which we will not concern ourselves with here).

The "interrogator" Rule

The function of the only rule in our ES is to find the question property structure facts, call **ask** to get an answer from the user, and then *assert* that answer as a fact in the knowledge base (also called the *fact-list*). This rule is used in all knowledge bases that use the interface and will call the **ask** function for all question property structure facts.

The Show Function

The **show** function provides a means to display text to the user. Its primary use in an ES is displaying results to the end-user. The text may be stored in a CLIPS fact or in a text file. In either case, **show** is passed text, which it parses into lines to fit in a display box on the screen. The user pages through the text until all has been shown.

Example 6. A fact and rule used to invoke **show**

```
(results show "The numerical rating of the site for use as a
dam site is: -100.  The ratings range from -100 to 100 with
100 being the best possible rating of a site for the construction
of a dam and reservoir.")

(defrule show-results
   (declare (salience -1000))
   (? show $?x)
   =>
   (show $?x))
```

Example 6 shows a rule and a fact that would *match* the conditions of the rule. Presumably the fact shown was created during the end-users consultation with the ES. The `(salience -1000)` would give the rule a low priority to fire, thus effectively holding the showing of results until the end of the consultation. The rest of the rule matches a condition with the fact, setting the variable `$?x` to the textual contents of the fact. The action statement, `(show $?x)`, calls the **show** function and passes the fact's contents.

The Title Function

Another accessory interface function is **title**, which can be passed five strings of text to be displayed as a title screen. The first four lines are centered and displayed in a box drawn on the screen, and the fifth allows for the optional display of a copyright note at the bottom of the title box.

Example 7. A fact and rule used to invoke title.

```
(dss-title title "DSS: Dam Site Selector" "Agricultural Engineering"
"Purdue University" "Bernie Engel  Dave Beasley"
"Copyright 1989 Purdue Research Foundation")

(defrule display-title
  (declare (salience 1000))
  (? title $?x)
  =>
  (title $?x))
```

Example 7 illustrates that the **title** function is used much the same as the **show** function. One difference is how title's text inputs are broken into separate strings, to indicate to the program what is to appear on each of the available title screen lines (Figure 2). The only other noticeable difference is *salience* which is set to *1000*, to insure that displaying the title screen is a high priority, since it should be the first thing the end-user sees.

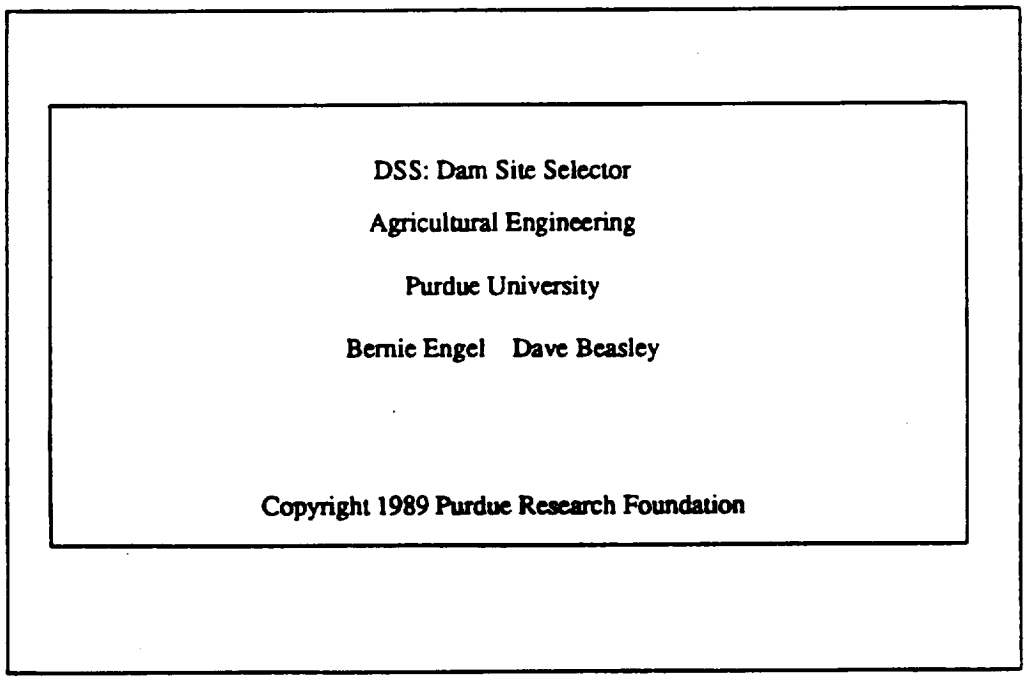

Figure 2. A title screen produced by Example 7.

Programming Notes

The development of the interface functions was done on a PC-AT, using the "C" language. The source code of the interface programs and CLIPS was *compiled* and *linked* together to make a customized *executable* CLIPS program. The executable program runs on IBM PC-compatible machines. Knowledge engineers may then develop ES using the customized CLIPS shell, making use of the additional functions **ask**, **show**, and **title**.

General Interface

A more general purpose version of the interface was developed by re-writing portions of the PC interface functions. The general purpose interface will work on any machine that runs CLIPS. As stated earlier, one of the reasons for moving to CLIPS was because of its ability to run on a wide variety of machines. The general interface version uses numbered menus with items selected by typing the number associated with the menu item. It does not allow the use of graphics nor does it use boxes around text as the PC version. CLIPS knowledge bases function identical for either interface, allowing applications to operate on a variety of machines.

Interface Application

The interface functions have been used in the development and conversion of several ES. Four of the ES that were transformed from commercial development/delivery tool formats into CLIPS are DAM SITE SELECTOR (DSS) [3], DOUBLE-CROP [4], MELON [5], and the GRAIN MARKETING ADVISOR (GMA) [6]. DAM SITE SELECTOR logically rates potential dam sites and provides an explanation of the factors influencing that rating. DOUBLE-CROP assists with the decision making processes in managing double crop soybeans following winter wheat. MELON assists muskmelon producers with proper management of their crop and with diagnosis and treatment of diseases. The GRAIN MARKETING ADVISOR assists grain producers in the selection of the appropriate grain marketing strategy for their situation. These knowledge bases in their original format required a commercial runtime tool to operate. After the transformation process, these knowledge bases run without a commercial tool and will run on a wider variety of computers. Minor information is lost in the transformation process, but other information is gained [7]. Additional details describing the knowledge base transformation process are provided by Engel et al. [7].

Conclusions

A PC-based end-user interface package has been created and integrated into the CLIPS ES development and run-time tool. CLIPS lacks an easy-to-use end-user interface development tool commonly found in many commercial ES development shells. The end-user interface development package has successfully been used to add interfaces to several CLIPS ES, in transformed knowledge bases, and in the development of new CLIPS ES. A similar set of C-based functions that are more general and thus will run on UNIX and other machines have also been developed and tested.

Benefits gained by using the parameter-driven interface package include:

- Less programming time is needed to complete the development of an application.
- Developers need not worry about many of the details of screen control or other output device-dependent problems.
- Uniformity and modularity is improved across the various programs developed that utilize the interface package.

References

1. Kendall, Kenneth, & Kendall, Julie 1988. Systems Analysis and Design. Prentice-Hall, Inc. Englewood Cliffs, NJ.

2. Faneud, Ross, & Kirk, Steven 1988. A UIMS for Building Metaphoric User Interfaces. In James A. Hender (ed.), Expert Systems: The User Interface, Norwood, NJ:Ablex.

3. Engel, B.A. and D.B. Beasley. 1988. DSS: A dam site selector expert system. In D. Hay (ed.), Planning Now for Irrigation and Drainage in the 21st Century, American Society of Civil Engineers, New York, New York. p. 553-560.

4. Halterman, S.T., J.R. Barrett, and M.L. Swearingin. (1988). *"Double Cropping Expert System"*, in the TRANSACTIONS of the ASAE, 31(1):234-239.

5. Latin, R., G.E. Miles, J.C. Rettinger, and J.R. Mitchell. (1989). *"An Expert System for Diagnosing Muskmelon Disorders"*, in Plant Disease, vol. 73.

6. Thieme, R.H., J.W. Uhrig, R.M. Peart, A.D. Whittaker, and J.R. Barrett. (1987). *"Expert System Techniques Applied to grain Marketing Analysis"*, in Computers and Electronics in Agriculture 1:299-308.

7. Engel, B.A., C. Baffaut, J.R. Barrett, J.B. Rogers, D.D. Jones. 1990. Knowledge transformation. Applied Artificial Intelligence 4:67-80.

Table 1. The ask function question property structure requirements.

Property Label	Uses/functions/requirements
prompt	The *prompt* label must always be followed by a string value, which is the question to be asked of the user.
expect	These are the alternative responses that may be provided for the user to choose from. Ask will present the alternatives in the form of a menu. If the *expect* label is not followed by alternatives, ask assumes it is not to construct a multiple-choice answer in the form of a menu, but instead will assume that the format will be user input of a number or text.
help	An optional string following this label is additional information about the topic of the question that may be of help to the user to understand the question or explain how it is to be answered. The *help* label may be followed by nothing, or the help text string.
why	An optional string following this label informs the user why the information requested by the question is important. The *why* label may be followed by nothing, or the why text in quotes.
value	This property is ignored by the ask function. However, a single-field value may be stored in the slot following this label.
value-type	When the *expect* property is followed by alternative answers, ask will prepare a menu from which the user may choose among the alternatives. The *value-type* property allows the knowledge engineer to indicate whether the given question may be answered by only one or more-than-one of the alternatives. The *value-type* property may also be used to indicate a numerical input is to be expected. Possible values for the *value-type* property are *SINGLEVALUED*, which indicates only one selection is allowed; *MULTIVALUED*, which means the user may choose one or more alternatives; and *NUMERIC*, meaning the user is to input a number. If *NUMERIC* is specified, there should be no *expect* values. If this property is left blank, *SINGLEVALUED* is assumed.
default	This property is ignored by the ask function. A single-field value for *default* may be stored in the slot following this label.
range	When the knowledge engineer wishes the user to enter a numerical answer to a question and needs to restrict the range of values the user may enter, the *range* property should be used. The values for this property should be two numbers separated by a blank. The ask function will require the user's answer to be between the two numbers. If *value-type* is *NUMERIC* and no range is set, ask will allow any number between -1000000000 and 1000000000. If the desired type of question is not to be numerical input, the *range* values must be blank. Also, no *expect* values are allowed in a question where numerical input is desired.
certainty-range	This property is ignored by the ask function, but the label *certainty-range* must be here. Two numerical values may be stored in slots following this label.
unknown	This property can be used if the knowledge engineer wishes *unknown* to be included as one of the menu alternatives. If followed by the value of *yes*, the option *unknown* will be added to list of alternatives.
gprompt	If the question is to use a graphics prompt, then this label should be followed by the file name of the image.
ghelp	If the question is to use a graphics help, then this label should be followed by the file name of the image.
gwhy	If the question is to use a graphics why, then this label should be followed by the file name of the image.