PRINCIPLES OF THE DEVELOPMENT OF INTERACTIVE QUERY EXPERT SYSTEMS

Valentin Kataev

***Abstract**: This paper describes the principles for the development of interactive query or question-answer expert systems (ES) in the Multi Studio software environment in Multi (the universal language), as well as in such environments as Visual Prolog and CLIPS. The paper also presents a comparative analysis of those principles checked by solving the test problems to show a significant advantage of Multi in the development of those ES according to the main quality parameters: language usability, work content of ES development, electronic memory size and speed of the developed ES.*

As a result of our study we formulate several principles of developing ES tool environments:

- *Development of a super-high-level universal environment language by integrating the best qualities of all the languages considered in the paper.*
- *Development of a universal structure of a knowledge base with a unitized syntax based on semantic networks.*
- *Development of a hybrid tool environment which can separately perform the following:*
    - *A one-time translation (compilation) of program and data input texts into an internal language of a knowledge base*
    - *Multiple fetch of programs from a knowledge base (the programs are executed by interpretation of those programs' instructions in a hybrid environment)*

***Keywords**: expert systems, CLIPS, Multi Studio, Prolog.*

***ACM Classification Keywords**: I.2.5 Programming Language and Software.*

Introduction

Being computer software QAES (ES), on the one hand, belong to data retrieval systems (ACM Computing Classification System (CCS) code: H.3.4) and, on the other hand, ES belong to decision-making systems (CCS: I.1.3).

Such systems are designed to send inquiries to a user online to receive certain answers from that person; as the required minimum of the answers is collected, the system generates special recommendations for the user what he or she is supposed to know or to execute.

The suggested recommendations (solutions), in general, can be informative, advising and also controlling. In the last case the question-answer expert system is built in a program-technical system as a control computing unit, when the information is input into the ES through a control equipment and then the ES outputs the generated controlling information to operating mechanisms.

The considered systems can be successfully applied in any range of human activity. A modern personal computer is quite a sufficient device to install and use rather advanced complex expert systems. But reality is far not so good.

The main reason is *complexity* of application development tools and maintenance of ES.

The present paper considers this issue by example of three toolsets applicable for development of information and advising systems including CLIPS, Multi Studio, and Visual Prolog.

Tools for ES Development

A standard software technology for developing ES is shown in Fig. 1.

Any ES is computer software. To develop it one always needs some other software (development environment) which converts algorithms of a certain developed ES into machine codes. There are two types of environments: compilers and interpreters.

The first ones develop stand-alone systems which can be run independently from their development environment. The second ones interpret input (source) language instructions i.e. they play the role of ES. The interpreters here can be subdivided into “pure” (i) and “hybrid” (ci). First the ci-environment translates (compiles) a text from the input (source) language into the internal language of the environment and saves the resulting structures in a knowledge base. This procedure is followed by a multiple execution (interpretive translation) of the environment internal instructions (i.e. ES functions are performed). Chart 1 demonstrates all of three versions of ES development.

Stand-alone ES(c):

- A program is securely built in the system
- Higher speed (despite for ES it does not matter as the most of computer time is “consumed” by a user).
- Complicated maintenance (every modification requires recompilation of programs).

- A compiling environment is, as a rule, universal with a universal classical language of a not very high level which requires a programmer of high qualification).

Integrated ES(i):

- A specially designed environment with a specialized language which can be of higher levels than that one of a compiling environment.
- Can execute an infinite number programs.
- Stand-alone data storage of input texts in the environment language in personal files of each program.
- Input texts are analyzed every time the ES is started.

Integrated ES(ci):

- As opposed to the environment mentioned above, the hybrid CI-environment first translates (compiles) the input (source) text into the internal language of the environment and saves the obtained structures into the knowledge base. Then this procedure is followed by a multiple execution (interpretive translation) of the environment internal instructions (i.e. ES functions are performed).
- The knowledge base is meant to store lots of programs and all the necessary data for them.
- The input language is preferably to be the universal higher-level language with a subset of instructions for effective creation and maintenance of ES. Otherwise, the use of the powerful system may appear to be improvident, wasteful and limited.
- Lower requirements of programmer skills or programming of ES can be performed by a knowledge engineer.

All other conditions being equal, we consider that the best variant is the hybrid environment. Below we give characteristics of three development systems.

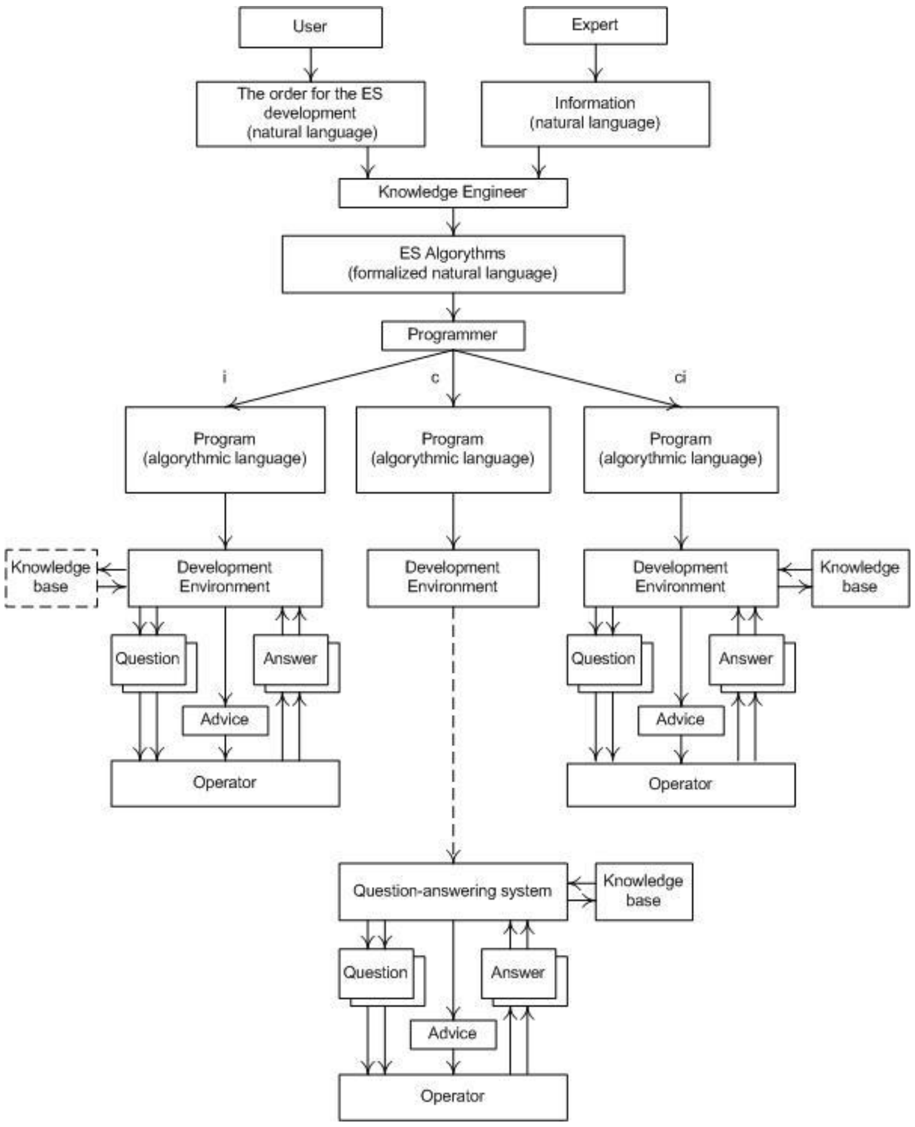
User
Expert
The order for the ES development (natural language)
Information (natural language)
Knowledge Engineer
ES Algorythms (formalized natural language)
Programmer
i
c
ci
Program (algorythmic language)
Program (algorythmic language)
Program (algorythmic language)
Knowledge base
Development Environment
Development Environment
Development Environment
Knowledge base
Question
Answer
Advice
Operator
Question
Answer
Advice
Operator
Question-answering system
Knowledge base
Question
Answer
Advice
Operator

Fig. 1. Versions of ES development

CLIPS Environment

CLIPS is a pure interpreter specially designed for developing expert systems.

Its basic developer is NASA (USA). The first version dates back to 1984. The last version is available since 6.30 2008.

Global language syntax: nested LISP-lists. Example: (X (+ 25 3)).

Data: facts (ordered and unordered). The last ones are represented by frames which consist of slots containing fact names and fact values. Example: Name (Bobby), year of birth (1997). The ordered facts do not contain names. In the list of facts they are ordered strictly by index. Variables can be of any type.

The program language is C-like.

Programs consist of modules which consist of rules and functions. The rules are arranged in a module in random order (sequence). The data exchange between modules is performed by functions "export" and "import", between the rules – by global variables. The rule basic structure: the module name, rule name, priority, formula. The rule formula: left side => right side.

The left side contains the conditions under which the program starts to execute the right side of the formula. Those conditions may include parameters for fact search and/or references to the rules which must precede the execution of this rule in the program stack. The right side of the rule may contain instructions and functions that implement the various actions. In particular, there may be queries to the PC operator, displayed in a dialog box, response message handlers and the output of the resulting recommendations for the operator to see. Also there may be the instruction of exit from the module (program).

A knowledge base is a set of text files containing data and programs in the source language. Each program must have its personal set of files.

Execution of a program includes three steps:

- CLIPS RAM emptying.
- Downloading text from files to RAM and dynamic forming of fact and rule lists.
- The program start and its execution (interpretation of the rules).

The program execution Technique.

The program performs sequential execution of the modules, beginning with the startup (MAIN). Inside the module, the start rule is the first to execute (this rule is automatically generated by the program itself). Then all the rules of the module are cyclically searched through in order to check the conditions to determine the feasibility of their execution. The selected rules are put in the operational queue in descending order of priority. If there are more than one rule of higher priority, then the program selects among them an only one according to the current conflict resolution strategy selected by the user (out of 7 possible strategies). The rule selected in such a way is to be activated and executed.

Then the process of selecting a new rule to be executed repeats. The module (program) execution is terminated if the next rule in turn ends with the "return" instruction or a new operational turn is empty.

An interface of the CLIPS system is very modest. A standard dialogue with the operator is performed in the main black-and-white window of the system in a command line mode.

Logging of ES execution is also performed in the main window.

Text coloring in windows (software or manual) is not supported. The system works with texts in Latin only, at least Version 6.30 (03/26/08).

As an example we take a small expert system for searching malfunctions in an automobile engine (AUTO) which can be found on a CD enclosed to [Джарратано, 2007].

As the program is pretty big, in Listing 2 we demonstrate just two rules with comments, and Listing 3 presents algorithms of this ES.

Listing 2. Two rules of AUTO program written in CLIPS.

```
(defrule determine-rotation-state ""  ; rule 1
   (working-state engine does-not-start) ; rule gets activated if the engine does not start
   (not (rotation-state engine ?)) ; engine shaft did not rotate
   (not (repair ?)) ; no solution (recommendation)
   =>
   (if (yes-or-no-p "Does the engine rotate (yes/no)? ") ; does the engine rotate?
      then
            (assert (rotation-state engine rotates)) ; if YES then the engine shaft rotates
            (assert (spark-state engine irregular-spark)) ; at that, the spark is irregular
      else
            (assert (rotation-state engine does-not-rotate)) ; if NO then the shaft does not rotate
            (assert (spark-state engine does-not-spark))) ; at that, the engine does not spark          )
(defrule determine-battery-state "" ; rule 2
    (rotation-state engine does-not-rotate) ; rule gets activated if the engine shaft
does not rotate
    (not (charge-state battery ?)) ; battery is not charged
    (not (repair ?)) ; no solution
    =>
    (if (yes-or-no-p "Is the battery charged (yes/no)? ") ; is the battery charged?
      then
          (assert (charge-state battery charged)) ; YES, the battery is charged
      else
          (assert (repair "Charge the battery.")) ; NO,   solution: Charge the
battery
          (assert (charge-state battery dead))) ; the battery is dead
    )
```

Listing 3. A Fragment of logging of ES in CLIPS.

```
    Does the engine rotate (yes/no)? n
    Is the battery charged (yes/no)? n
    Suggested Repair:
    Charge the battery.
    CLIPS>
```

Visual Prolog

Visual Prolog is an interpreter [Адаменко и др., 2003].

Basic developer: Prolog Development Center (Denmark), since 1984. (Turbo Prolog => Prolog PDC => Visual Prolog).

Global syntax: the list of sentences.

The Prolog program includes the following sections: DOMAINS, PREDICATES, CLAUSES, GOAL GOAL contains a list of target predicates. The section named PREDICATES contains predicate declarations (names and types of arguments), DOMAINS specifies nonstandard (custom) types of predicate arguments (predicate variables). CLAUSES contains detailed descriptions of facts and rules.

Facts are attitudes or properties that are always "true."

Example of properties: ***red*** (apple). Attitude: ***like*** (anne apple).

Rule is a property or Attitude, which is true, if a number of other relations are true.

Rule Format: <fact>: - body <list of truth conditions for the fact>

Example: ***like*** (anne Fruit): **-** ***red*** (Fruit).

The program execution technique. The program searches for a solution by inference, starting with the first rule, and scanning through the entire list of sentences to determine which sentence should follow the current one. At that, the program selects conditions from the body of the current rule satisfying the conditions of other rules. If all conditions are true, then the rule header is considered to be true and the goal is accomplished. Otherwise, the rule is skipped (goal is not accomplished).

The Visual Prolog interface represents a multi-document window. A text of a separate program is output in a separate window (document). Keywords, text literals, comments, variables, etc. are coloured. Errors found in the course of compilation are output in a separate window. The development language is English. In case the compilation process was successful, the software displays a special window representing a command line.

In Listing 4 we can see the rules from Listing 2 (in Prolog), and in Listing 5 – a log of their execution.

Listing 4. Two rules from ES in Prolog

```
PREDICATES
nondeterm result; nondeterm battery; contacts; ignition_coil
CLAUSES
result      :- write("\nDoes the engine rotate?"),       readchar(Answer),
Answer = 'y'                                              /*if (result)*/
   ,!, contacts.                                          /* then (result)*/
result      :- battery.                                   /* else (result)*/
contacts:- write(" yes\nRun 'Contacts' ").
battery     :- write(" no\nIs the battery charged?"),     readchar(Answer),
Answer = 'y'                                              /*if (battery)*/
   ,!, ignition_coil.                                     /* then (battery)*/
battery :- write(" no\nCHARGE THE BATTERY ").             /* else (battery)*/
ignition_coil         :- write(" yes\nRun 'Ignition_coil' ").
GOAL
result, write("\n\nSolution is found").
```

Listing 5. A Fragment of logging of ES in Prolog

```
Does the engine rotate? n
```

Is the battery charged? n
CHARGE THE BATTERY

Multi Studio

Multi Studio is a hybrid interpreter. Version 082ev Engl (2012).

Basic developer: Center of Intelligent Technologies (Russia), since 2007 [Катаев, 2012].

Global language syntax: nested functions and procedures. Example: ***X (+ (25 3))***.

The logical form and physical structure of knowledge representation (both data and programs) are universal. These are syntactic networks, on which semantic subnets are superimposed. Each network node has a "semant" - a word of Multi, which can be a number, text line, date, name, computer instruction... Instruction is a function (procedure) or its argument. Examples of instructions: * (product) ***sin.*** (sine) ***P#*** (arguments alternated with spaces are output onto the computer monitor).

The program is a semantic subnetwork. In general, it includes modules and sentences. A module has a name in the high node and may also consist of modules and sentences. A sentence is a part of the program, finished by '***;***'. The program itself can be a module. The module can consist of a single sentence. Example: ***ZZ (12 34);***

When the source text is input into the system, all the modules are recorded in the knowledge base (container) and are linked in a single network by name. When the program is run it moves along the nodes "from top to bottom and from left to right." This navigation can be dynamically changed by some of instructions ("cycle", "or" ...), as well as by getting a negative result while executing an instruction.

In this language there are several instructions that allow a user to easily develop a query-response (question-answer) system. The principal one among them is ***menu.***; it has the following structure:

menu. ("Header" Option 1 Option 2 ...)

The option structure: field_name (actions)

The field may be a name, text, number, instruction.

The header can be a statement or question of the program to the user. Variants are possible answers (responses) of the user and actions which the program will perform in case this or that option is selected. In case there action list is empty the program goes back to the loop which is closest, and if there are no loops to execute anymore the program ends.

The instruction opens a menu dialog window with the header, fields of options and additional system fields. As an option (answer) is selected the corresponding action is executed. In particular, that action may be moving to the module which opens a new menu window.

A special case of options is Help. The Help field is a text with an exclamation mark as the very first character. Selecting the Help field opens files specified in the instruction. After the help files are closed, the program automatically returns to the current menu.

Clicking on the additional fields allow the user to return to the previous menu, exit the current loop or quit the program.

The interface of Multi Studio is very diverse, including

- color highlighting of input text,
- syntax error indication (red font color) in the text, up to a single symbol,
- colorful navigation about the program and data during program testing,
- computation result insonification in the human language
- eight system windows with lots of tabs displaying
  - the input text of the program,
  - data under process,
  - the program log,
  - intermediate and final results of its execution
  - help-windows

In addition to the system windows the user program can generate a variety of custom (user) windows (including the form of complex configurations, and menu dialog windows mentioned above).

Multi Studio deals with texts in different natural languages (including Russian). Instructions of Multi are mostly symbolically notated, but the user of Multi Studio can easily change the notation system of by introducing into the dictionary additional synonym instructions.

Example: Listing 6 shows a fragment of the two rules in Multi, which are demonstrated in Listing 2 in terms of CLIPS. Fig. 2 shows the dialog boxes activated by those rules. Listing 8 presents a fragment of the ES log in the Multi Studio environment.

Listing 6. Two rules of ES AUTO in Multi

```
Engine_does_not_start ( menu. ( "Does the engine rotate?"
"yes"(Contacts)                  // if YES then execute "Contacts" module
"no"( Battery);                  // if NO then execute " Battery " module
```

```
Battery ( menu.(  "Is the battery charged?"
"yes"( ignition_coil)                    // if YES then execute "ignition_coil "
"no                                      // if NO then
CHARGE THE BATTERY";
```

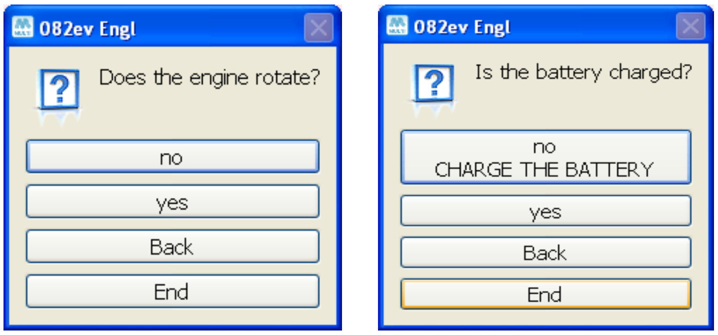

Fig. 2. Dialog boxes of the first and second rules

Listing 8. Fragment of the ES log in Multi Studio environment (two rules)

```
Does the engine rotate?  no
Is the battery charged?  no
CHARGE THE BATTERY
```

Conclusion

The Multi language has the following advantages: universal syntax, higher language level, easiness and conciseness.

As opposed to CLIPS and Prolog, MS is a hybrid interpreter, so MS is faster, more reliable and can process bigger amounts of data and execute big programs. Networks of programs and data get formed as a one-time start compilation is executed, the program forms networks of programs (internal representation), where each rule is linked with the corresponding rules. As a result, when the ES is run the program performs one-time selection of the appropriate rules from the beginning to the decision from the top down, which increases the performance speed (selection speed). CLIPS and Prolog perform a multiple looping searching the rules able to continue the chain. Software performs bottom-up selection of rules and the one-time linking. At that, a conflict may emerge when more than one rule is selected. There are special strategies to be used to solve this trouble.

Besides, Multi allows the user to perform a reverse translation from the internal language to the different national languages (with appropriate dictionaries available which can be created by the user).

The MS interface is more developed compared with the interfaces of CLIPS and Visual Prolog.

Based on the considered ES development environments, we can recommend the development of tool environments in the following directions: development of hybrid systems with a super-high-level universal language and integration of the best qualities of all the languages mentioned above.

Bibliography

[Адаменко и др., 2003] Логическое программирование и Visual Prolog /Адаменко А.Н. [и др.]; - СПб.: БХВ-Петербург, 2003.

[Джарратано, 2007] Джарратано, Джозеф. Экспертные системы: принципы разработки и программирование, 4-е издание. Пер. с англ. / Джарратано, Джозеф, Райли, Гари ; – М. : ООО "И.Д. Вильямс", 2007.

[Катаев, 2012] Катаев, В.А. Разработка экспертных систем в среде Multi Studio / В.А.Катаев // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2012): материалы II Междунар. научн-техн. конф. (Минск, 16–18 февраля 2012 г.); – Минск: БГУИР, 2012, С. 207–212.

Authors' Information

Valentin Kataev *–Intellectual Systems Laboratory Chief, LtD "Perm Scientific Industrial Instrument-Making Company", Perm, Russia; e-mail: bravo666666@yandex.ru*

Major Fields of Scientific Research: Artificial Intellect, Computational linguistics.