KNOWLEDGE-BASED SYSTENS DEVELOPMENT TOOLS EVALUATION: EDUCATIONAL POINT OF VIEW *

Mária Bieliková, Pavol Návrat
Slovak University of Technology, Dept. of Computer Science and Engineering,
Ilkovičova 3, 812 19 Bratislava, Slovakia
{bielik, navrat}@elf.stuba.sk
http://www.dcs.elf.stuba.sk/~bielik, http://www.elf.stuba.sk/~navrat

Abstract: *Languages, tools and environments play a significant role in a knowledge-based system development life cycle. This paper presents an approach to training knowledge engineers together with evaluation of languages used for knowledge-based system development from the educational point of view. An experiment aimed to this evaluation is described. We claim that general programming languages can also be used in training. We provide experimental evaluation of their varying degree of suitability with respect to differences of languages.*

Keywords: knowledge-based systems, knowledge representation, development tool

1 Introduction

The amount of information that a human brain can effectively store, retrieve and process seems to be relatively limited. The experience shows that one's memory does not always react quickly and precisely to access calls for information. Thus the need emerged to develop systems that allow knowledge to be manipulated. Response to this need is a long time effort in the field of artificial intelligence, in particular of knowledge engineering. Knowledge engineering can be characterized as a process of building a knowledge base (Russel, Norvig, 1995). A knowledge engineer is someone who investigates a particular domain, determines what concepts are important in that domain, and creates a formal representation of the objects and relations in the domain.

Applications of today's knowledge-based systems (and of expert systems in particular) cover a very broad area of a human expertise. One promising area of their use is software development automation (Tichy, 1987; Návrat, Rozinajová, 1996; Návrat, 1996). Although some people argue that the golden age of knowledge-based systems is over, there is a strong evidence of the growth of knowledge-based systems applications (Hayes-Roth, Jacobstein, 1994). This requires more of experienced knowledge engineers in the development of new and more sophisticated systems. To face the lack of experienced knowledge engineers, new and more effective methods are to be used to educate them. Their training should include gaining experience in building intelligent applications.

From several points of view, the existing systems provide an excellent educational basis for knowledge engineering. Despite this fact, however, most of them lack certain features that we find desirable when used to train prospective knowledge engineers. We believe that knowledge engineers are expected to understand more deeply the nature of

*The work reported here was partially supported by Slovak Science Grant Agency, *Methods and Tools for Development of Software Systems.*

the knowledge representation and reasoning processes than users of an expert system. Similarly, computer users who ocasionally write programs do not need to understand details of the compilation process, but professional programmers and software engineers definitely should. As a consequence, compiler design is part of a body of knowledge of software engineering (Ford, 1991) and a traditional subject in software related curricula. It is quite common to assign students to write at least some part of a simple compiler. It is understood that this is important despite the fact that most of the students shall probably never write a full and complete compiler in their career. On the other hand, it is certain that most of them shall very often need to program some sort or a translation from one language of data representation notation to another because the concepts of language and translation are very general.

In knowledge engineering, it is important to gain a deeper insight into the internal structure and functionality of knowledge-based systems. The problem with most of the systems that are available is that their functionality is hard-wired and almost no experiments with adjusting their reasoning methods can be accomplished. Having the possibility to design own enhancements to the knowledge-based system and incorporate them to the existing shell rises the level of experience and expertise of future knowledge engineers.

At the Slovak University of Technology, for several years we have been using in the *Knowledge-Based Systems* master course an expert system shell KEX developed at the department (Bieliková *et al.*, 1990) and an expert system development tool NEXPERT that is commercially available. Our teaching approach has been to let our students experiment with not only developing knowledge-based system applications, but developing, adjusting, enhancing (parts of) a knowledge-based system development tool as well. To accomplish the latter task, assignments to modify KEX and assignments to develop parts of a knowledge-based system development tool in Lisp or Prolog were solved.

Nowadays, fast evolution of programming languages and supporting environments with still higher level of abstraction make suitable not only the "traditional" AI languages such as Lisp or Prolog, but also some non-AI languages for knowledge-based systems development. Therefore, we decided to modify the teaching approach and to let our students use such languages in developing (parts of) a simple knowledge-based system shell, too.

Our intention was to have our students experience that AI languages and expert systems shells are more powerful for knowledge-based systems development than non-AI languages. Results achieved showed to us that it is possible for the students to undertake an effective knowledge-based system development by using also other tools than such specialized tools such as expert system shells or special AI languages. This supports the claim that also certain other general-purpose programming languages can be used. The experience shows that this is the case especially for an implementation of the knowledge representation and control structures needed for knowledge interpretation.

The paper is organized as follows. In Section 2, the approach to knowledge-based systems education together with an experiment aiming to compare different tools in knowledge-based system life cycle are described. The tools used are briefly characterized. Next, we give some examples of problems solved by means of different tools. In

Section 3, we discuss results achieved and compare tools used. The paper closes with our conclusions.

2 Knowledge-based system development

Because a knowledge-based system is a software system (that manipulates encoded knowledge to solve a problem), conventional software life cycle models should be considered (Boehm, 1988). In particular, they seem to be applicable to development of an inference engine, or a user interface together with supporting features such as editor, explanation facility, etc. On the other hand, specialized life cycles for knowledge-based systems were proposed (Agarwal, Tanniru, 1990; Trenouth, 1991). They are designed mainly to knowledge base life cycle development which is just one (although very important) component of a knowledge-based system.

In training of knowledge engineers we try to cover these both aspects of knowledge-based systems development. Although stress in the Knowledge-Based Systems course is put on the knowledge representation, the students are trained in knowledge acquisition, too. Availability of relevant experts is achieved by either aproaching experts whenever possible (be it a parent, friend, etc.) or by simulating experts by assuming that appropriate students act as "experts" in areas they have mastered sufficiently well in their previous education and practice.

The objective of knowledge representation is to express knowledge in computer tractable form, such that it can be used for problem solving. As we mentioned earlier for several years we used expert system shell KEX together with its knowledge representation formalism, KEX/L language for expert systems development teaching (Frič *et al.*, 1993).

There is a strong evidence of the growth of new high level programming languages and environments which exploit several features used by knowledge representation formalisms. We decided to test their suitability for knowledge-based systems development. Our main interest is to compare the development supported by programming tools with the development using AI programming languages or an expert system shell.

Our experiment was designed as follows:

- The expert system development life cycle consisting of (i) knowledge acquisition, (ii) knowledge structuralization and conceptualization, (iii) formalization, (iv) implementation and (v) testing phases was adopted.

- Knowledge acquisition, structuralization and conceptualization were performed independently of the supporting environment (or tools) and knowledge representation formalism chosen.

- Formalization and implementation were performed by each student twice. First, they used an arbitrary general purpose programming language. Then, they used the expert system shell KEX.

- Knowledge-based systems prototypes that had been developed were compared and evaluated.

We will look now a little more deeply into the knowledge representation languages used. Russell and Norvig (Russel, Norvig, 1995) characterize a good knowledge representation language by the following properties: "It should be expressive and concise so that we can say everything we need to say succinctly. It should be unambiguous and independent of context, so that what we say today will still be interpretable tomorrow. And it should be effective in the sense that there should be an inference procedure that can make new inferences from sentences in our language."

To achieve these properties of a good knowledge representation by implementing knowledge-based system in some programming language, interpretation of knowledge representation formalism chosen should be used as a method of implementation. Direct use of a programming language is possible only in very specialized areas and even this at the expense of loosing the fundamental property of knowledge-based systems: separation of knowledge from their interpretation (reasoning, inference). This property allows the knowledge engineer to worry only about the content of the knowledge, and not about how it will be used by an inference procedure.

2.1 Programming languages used in the experiment

Choice of a programming language for a first prototype was left to the students. Our intention was to cover as broad a spectrum of languages as possible and at the same time to exploit familiarity with the language for an effective development.

Programming languages that were actually used can be divided to four main groups:

1. procedural languages (C, Pascal),
2. declarative languages (Lisp, Prolog),
3. object-oriented languages (C++),
4. fourth generation languages – 4GL (MS Access, Magic, Power Builder, Visual C).

This relatively broad spectrum of languages mirrors current state in programming independently of the application being developed. From the knowledge-based system development point of view, a level of maturity of the language's supporting environment is also important. Good debugging facilities, a good editor, an ability to retrace one's steps and recover from mistakes are required.

2.2 The KEX tool and knowledge representation

The KEX tool is an expert systems shell which incorporates several support tools: knowledge base editor, interface designer, debugger, and database interface. The knowledge representation formalism has main impact on the applicability of particular expert system shell in various problem domains.

The *KEX/L language* is a hybrid formalism that combines frames, rules, and procedural extensions. The use of an object-oriented approach provides a unifying testbed for fusion of all above mentioned schemes. The frame formalism used in KEX/L represents the object-oriented extension of frames which incorporates the concept of frames.

Another object-oriented feature is introduced at the attribute level where generic attributes are allowed. Rules in KEX/L are structured into rulesets. The rule formalism in KEX/L allows to represent specific user-defined conflict resolution strategies and also to alter the actual conflict resolution strategy during the problem solving process. Specialized conflict resolution strategies can be assigned to every ruleset. Procedural knowledge can be represented by functions and procedures in a language similar to Pascal.

The system architecture was designed to be very flexible to support future enhancements and experiments. Precisely defined interfaces between modules create a basis for futher experiments with the system architecture and with implementation of separate modules. By using the above mentioned approach, the system architecture can be easily restructured according to the specific requirements resulting from practical experiments performed by the students. Replacement and reimplementation of single modules provide a good training facility for getting deeper insight into functionality of individual modules.

2.3 Examples of developed applications

Students involved in the experiment solved different problems in several domains such as medicine, engineering (electronics circuits testing, motor repair), computer science (computer configuration), business (businessman advisor), mathematics (differential equations solving), psychology (selection of the most suitable husband/wife for a particular person). When problem categories are considered, the applications fall into the following groups:

- diagnosis – medical and technical,
- interpretation,
- repair,
- design (restricted to a known set of possible solutions).

Most of the problems that were solved as projects by the students cover the group of diagnostic problems. The main reason is that one semester is usually a too short period to develop more complicated application that would cover also complicated design-type problems. On the other hand, our experience is that even the described kind of training is suitable for future knowledge engineers.

Three main models of problem solution were used by students: (i) sequential evaluation of symptoms in one step, (ii) multi-step evaluation of symptoms, (iii) tree model of the problem solution. These models were often extended with uncertainty, marking solutions, or elimination of impossible solutions, and implemented in some combination.

3 Results achieved and Discussion

Artificial intelligence programming is inherently an exploratory one. The program is often a vehicle through which we explore the problem domain and discover solution strategies. A language suitable for artificial intelligence programming should therefore provide the following features: modularity, extensibility, useful high-level constructs,

Characteristics	Procedural languages	Declarative languages	OO languages	4GL	Expert system shell
Knowledge representation	– data structures – control structures of the language	– functions/ predicates – list structures	– classes, objects	– relational database	– rules, frames, procedures
Development support	high	medium	high	very high	medium
Explanation facility	not supported	not supported	not supported	not supported	supported at the syntactical level
Change of the solution set	– add new condition or – add new structure	– add new function/ predicate	– add new class	– add new record to the database	– add new rule and/or frame
Change of the solution characteristics (attributes)	– difficult if knowledge are represented as control structures	– modify few functions/ predicates	– modify class and few objects	– add new attribute	– add new ruleset, modify few rules
Change of the solution model	– new prototype should be created	– some of the functions/ predicates could be reused	– flexible only for tree method	– new prototype should be created	– some of the knowledge could be reused
Development effort	medium	medium	low to medium	low	low

Table 1: Knowledge-based system development tools comparison

support of early prototyping, program readability, interpreted and compiled modes (Luger, Stubblefield, 1993).

A similar view of suitability of programming language (or tool) for exploratory development can be found in (Trenouth, 1991). Four principles are considered: (i) principle of execution (availability of executable intermediate products), (ii) principle of extension (easy modification in order to exihibit new behaviour without adversely affecting existing behaviour), (iii) principle of exploration of software alternatives (previous project states should be interactively recoverable), (iv) principle of explanation (analysis and visualisation of life-cycle).

In most literature, Lisp and Prolog are considered to be the implementation languages (at least as the language of the first prototypical implementation). On the other hand, our experiment showed us that limitations of currently available programming languages from the knowledge-based systems point of view are not so big as they were before or may still seem today.

Table 1 shows some of the results obtained from our experiment. Problems solved were briefly introduced above. Their common feature is that problems have finite set of solutions which is known before process of problem solution is initiated. Process of solution could be then considered as the solution selection based on the problem characteristics description (e.g. disease symptoms). Three aspects of change are considered: change of the solution set, change of the solution characteristics and change of the solution model (sequential, multi-step or tree).

The outcome of procedural languages (C and Pascal in the experiment) is not very surprising. A prototype development for simple problems can be effective mainly due to the fact that the procedural programming skills are frequently already available (no cost) and the cost of mastering a knowledge representation language is relatively high. Their unsuitability for exploratory programming is evident for more complex problems. Separation of knowledge from its interpretation is not easy, cost of future modifications, which is a primary principle of exploratory programming, is high.

We are not going to comment on the AI languages such as Lisp and Prolog. Their suitability for knowledge-based systems development was evident in the experiment and this fact is largely accepted by the programming community. In particular, Prolog itself can serve as a primitive shell with a default inference method already implemented.

Object-oriented languages together with four generation languages make programming easier mainly by providing a better support to the re-use of software components.

Comparison with the expert system shell from the development effort evaluation is not fair because knowledge control is implemented already. We include the column mainly for the sake of completness of the record of our experiments.

4 Conclusion

Knowledge-based systems can be constructed with the aid of a large number of tools. The tools range from programming languages to integrated environments. Consideration of their suitability is based on the knowledge-based systems life cycle which is exploratory in its nature. We claim that set of languages applicable to knowledge-based systems development is growing and subsumes at least object-oriented languages and fourth generation languages.

Our experiment which was aimed at evaluating current languages with respect to their suitability for knowledge-based systems development came out as a very good education tool. Students developed expert system in their favourite language which is very often considered by them as "the best, the most effective and the most suitable for all kinds of problems". They were not restricted in an approach, e.g. knowledge base and inference engine did not have to be separated. After that they explored advantages of the expert system shell and some of them (mainly "native" C and Pascal programmers) were disapointed by the fact that their favourite language is not as best as they believed. More importantly, they realised how important are the principles of development of knowledge-based systems for their successful implementation.

5 References

Agarwal, R. and Tanniru, M. (1990), "Systems development life-cycle for expert systems", *Knowledge-Based Systems*, Vol. 3, No. 3, pp. 170–180.

Bieliková, M., Frič, P., Galbavý, M. and Vojtek, V. (1992), "KEX – an environment for development of expert systems", *Proc. 14th Int. Conf. on Information Technology Interfaces ITI'92*, Pula, 1992, pp. 153–158.

Boehm, B.W. (1988), "A spiral model of software development and enhancement", *IEEE Computer*, Vol. 21, No. 5, pp. 61–72.

Ford, G. (1991), "SEI report on graduate software engineering education", Technical Report CMU/SEI-91-TR-2, CMU Software Engineering Institute.

Frič, P., Bieliková, M. and Kapustík, I. (1993), "An environment for knowledge engineering education support", *Proc. of AI-ED 93*, ed. by P. Brna, S. Ohlsson, and H. Pain, AACE, 1993, pp. 557.

Hayes-Roth, F. and Jacobstein, N. (1994), "The state of knowledge-based systems", *Communications of the ACM*, Vol. 37, No.3, pp. 27–39.

Luger, G.F. and Stubblefield, W.A. (1993), *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, The Benjamin/Cummings Publ. Company, Inc., 2 edition.

Návrat, P. (1996), "What is the knowledge that knowledge based programming is based on?: An analysis", *Proc. 8th Annual Workshop Psychology of Programming*, Gent, 1996, ed. by P. Vanneste, K. Bertels, B. De Decker, and J.M. Jaques, pp. 95–104.

Návrat, P. and Rozinajová, V. (1996), "Knowledge based programming: An experiment in selecting a data type", *Arab Gulf Journal of Scientific Research*, Vol. 14, No. 1, pp. 79–100.

Russell, S.J. and Norvig, P. (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Tichy, W.F. (1987), "What can software engineers learn from artificial intelligence", *IEEE Computer*, Nov., pp. 43–54.

Trenouth, J. (1991), "A survey of exploratory software development", *The Computer Journal*, Vol. 34, No. 2, pp. 153–163.