Transactions on Information and Communications Technologies vol 16, © 1996 WIT Press, www.witpress.com, ISSN 1743-3517

Technical criteria for selecting expert system shells for engineering applications

Angel Pasqual del Pobil* and Valentín Arroyo**
**Department of Informatics, Jaume-I University, Penyeta Roja Campus, E-12071 Castellón, Spain, EMail: pobil@inf.uji.es*
***E.T.S. Ing. de Caminos, C. y P., University of Cantabria, Av. de los Castros, s/n, E-39071 Santander, Spain*
EMail: valentin@fltq.es

Abstract

A large number of Artificial Intelligence applications in engineering are based on the use of expert systems. One of the fundamental decisions that should be made initially is the selection of the most appropriate development shell. The most frequent types of engineering applications has evolved over the last few years in such a way that it is no longer possible to think of a single large isolated expert system running on a mainframe or minicomputer; instead, we must now consider an embedded subsystem subject to new requirements. The existence of more than a hundred commercial shells currently available on the market makes the problem even more complicated. The bases for an initial evaluation and selection of these shells for the most frequent type of engineering problems are presented in this paper. It is intended that the different aspects involved in the problem will be clarified. To achieve this, a systematic and rigorous process based on an empirical methodology for expert system shells evaluation and selection has been followed. In the paper, a general framework for the problem is first presented and then the identification of the capabilities and characteristics of the required shell are described: user aspects, technical aspects, cost and vendor. Next, the requirements imposed by the application type are discussed: eight application types are distinguished, the requirements for reasoning mode and the problem of fact-base unfeasibility are dealt with. The question of rapid prototype development is also analyzed. Some capacities that are critical for the success of the project are identified, such as the support for personal computers, embeddability, backward chaining, connection to databases, etc. Following the established criteria, details of 31 common shells are finally included.

1 Introduction

Knowledge-based systems or expert systems are, in general, one of the most successfully applied Artificial Intelligence techniques, especially in engineering. A firm often considers using an expert system to deal with a particular engineering problem. An initial and fundamental decision that should be made is the selection of the most appropriate tool or development shell. The most frequent type of engineering applications has evolved over the last few years in such a way that it is no longer possible to think of a single large isolated expert system; we must now think of a subsystem which becomes embedded in the rest of the firm's computer systems. (We shall call this *the home system*). Furthermore, new key requirements for evaluating expert system viability have appeared.

Since there is a stress on the engineering project's income-yield capacity and success, this paper gives an initial evaluation and selection of expert system development shells or tools. Unlike software evaluation which has been studied thoroughly by software engineering, in the case of expert systems —due to their special characteristics— the normal evaluation techniques are not valid, thus often leading to selections made in an inadequate manner. This paper intends to clarify the different aspects involved in the problem. To achieve this, a systematic and rigorous process based on an empirical methodology for expert system shells evaluation and selection has been followed.

When the feasibility of including Artificial Intelligence techniques is considered, certain aspects of the problem seem to suggest the necessity of answering the following questions:

- Is an Expert System viable for the problem in question?

- Which shell would be the most appropriate for its development?

The second question is conditioned by an affirmative answer to the first and it refers to the selection of the expert system shell. Due to space limitations, this paper assumes that the problem has been previously analyzed and that the requirements have already been specified. Likewise, it is supposed that the viability of an expert system-based solution has been studied. More details about these points can be found in [19]. This paper deals with the evaluation and selection of the most appropriate tool or ES shell.

2 General Framework for the Problem

Before trying to answer the question of shell selection, it is necessary to define the fundamental characteristics of the type of problem to be dealt with and the minimum requirements we must demand to the system. As we shall see, the nature of the problem will condition the answer to the question which this paper discusses.

The home system will be based on the massive generation and handling of information. The Expert System's mission will be that of a support mechanism in interpreting this large volume of information in order to help in the decision-making process by suggesting actions to correct malfunctions.

All computer science projects entail the previous preparation of a *requirements specification* document. We can identify some of these objectives and restrictions:

- **User Profile**. The final system user cannot be expected to have a vast knowledge of the system from a computer science standpoint, although he will be well-versed in the problem area. The system should be friendly and easy to use giving extensive information and explanations about its results.
- **System Output**. In accordance with the user profile, it should have a graphic user interface, the possibility of showing results graphically, and the embeddement in the home system as well as connections with other systems in a way that is transparent to the user. This person will be able to ask the system for additional information or explanations about its functioning. The system will also be able to ask the user for any relevant information which is not available.
- **Hardware Constraints**. The expert system should function on a distributed network of personal computers (preferably with a client/server architecture).
- **External Interface Constraints**. The expert system should be completely embedded in the home system and it should have direct access to the home system's databases.

3 ES Shell Evaluation and Selection

ES shell selection is a critical question: using an inappropriate shell for a particular application is like walking a great distance wearing shoes that are several sizes too small. The resulting system would not be effective and would even make the project fail. Expert system shell evaluation and selection are difficult due to the lack of industrial standards and the rapid pace at which this technology is developing. This is made even more difficult by the existence of over a hundred commercial shells currently available on the market —widely varied in terms of quality and price— and by the habitual use of inconsistent terminology in the information provided by vendors.

Shell evaluation and selection must go through the following steps:

1. The identification of the capabilities and characteristics the shell requires.
2. The identification of a reduced set of shell candidates.
3. The evaluation of shell candidates based on identified requirements.
4. The selection of the most appropriate shell.

This paper mainly covers the first two/three stages mentioned above in which an initial *screening* is done based on critical requirements. Final evaluation and selection require dealing with concrete shells for a concrete problem.

Four main sources exist which give us the requirements that we should ask for in all cases of commercial expert system shells.

- application type
- technical environment
- human factors
- economic factors

We have already mentioned some human and technical factors when discussing requirements specification (section 2). We will comment on application type, given its importance, in section 3.2. The basic criterion for selection will be to compare the different requirements with the capabilities that each shell offers. These capabilities can also be grouped in a similar way:

- user aspects
- cost
- technical characteristics
- vendor

3.1 Expert System Shell Capabilities

Within the four categories stated above we can single out numerous capabilities [26,27] which we shall briefly comment on in this section. Only the ones

considered to be most important to our case have been included. The ones that are critical for the success of the project —which will be briefly justified— have been highlighted in bold face.

a) User aspects

- **Final user interface**. This refers to the capabilities that will have the final product for its ordinary use.
  - **Explanation facility**
  - Graphic results
  - Tutorial
  - Windows and menus
  - **Complete and intelligible documentation**

As users will not tend to possess a specialized knowledge of either computer science or artificial intelligence, it will be critical that the capabilities allow the user to feel comfortable and to be able to ask the system for additional explanations.

- **Development interface**. This refers to the capabilities that the system offers during its developmental stage. The user is now the knowledge engineer and his/her team.
  - **Documentation**
  - Graphics
  - Tutorial
  - Windows and menus
  - **Explanation facility**
  - **Fast prototype development**
  - Editing and debugging tools: **inference trace**, etc.

The development team is usually specialized in computer science, but not in Artificial Intelligence or expert systems. For that reason, the system should be easy to use and it should not be just for specialists. Helpful assistance for debugging will be essential in the cycles of system verification/validation. The capability for rapid prototype development will be discussed further ahead.

b) Technical characteristics

- **System interface**. These capabilities are relative to the hardware and language on which the system is implemented.
  - Portability
  - **Embeddability**
  - **Support for personal computers**
  - Support for multiusers

Each shell is available for a limited number of hardware platforms. Support for PCs will be a critical question. On the other hand, the base language may or

may not allow the system to become embedded as part of the complete home system in such a way that it can co-exist and cooperate with it. This will be generally impossible if the base language is Lisp.

- **Inference Engine**. This refers to the heart of the expert system, that which does the reasoning. Shells usually offer one of the two possible chaining modes or control strategies:

  - Forward chaining
  - **Backward chaining**

  Other factors that should be kept in mind are:

  - Rule priority assignment
  - Certainty factors

  We base the justification for the need of backward chaining on the application type. This will be analyzed in section 3.2.

- **Knowledge base**. This refers to the way of representing knowledge according to different possible techniques:

  - **Production Rules**
  - Frames
  - Partitioning rules into sets
  - Inheritance mechanisms

  For an initial application the usual production rules may be enough, although the different shells vary a great deal in the richness and flexibility of rules. The use of frames and inheritance would be left for a later stage of system extension.

- **Data interface**. This is relative to the system's capability to access external software and to permit its interaction with other systems that make up the firm's computer environment.

  - **Connection to databases**
  - Access to support language
  - Connection to special purpose software: spreadsheets, etc.

  This is a fundamental capability that many shells do not offer. It allows the incorporation of capabilities that the shell does not include by accessing directly to them: for example, for showing spreadsheet data in the form of a graph, etc. In the home system, it is usual to find all relevant information stored in the databases; thus, access to them will be of critical importance. It will be necessary, therefore, to know the home system's concrete databases before making the final shell selection.

c) Cost

The prices of expert system development shells range from $0 (public domain software) to $120,000 (for large mainframes). The prices for personal computers range from $100 to $5,000. The size of the computer on which the

shell functions will naturally tend to make the price go up. Nevertheless, sometimes there does not seem to exist any logical relationship between quality and price for comparable hardware platforms.

d) Vendor

Vendor support is important: for example, if the vendor offers training and consulting. Moreover, the firm's experience in the expert system market or computer market in general will allow it to assess the course of development of previous products that have or have not been successful.

3.2 Requirements Imposed by the Application Type

In specialized literature it is widely agreed that different application types require different capability types for expert system shells [5, 7, 10, 12, 23]. In a study done by the University of North Carolina [28], 36 different application types were distinguished. Among these applications, the following are relevant to our case:

- **Monitoring.** To observe a system while it is functioning and to give a warning when it behaves in an unexpected or unusual way according to the foreseen model.
- **Interpretation**. To select a hypothesis based on data obtained from measurements and information deduced from them.
- **Diagnosis**. To determine the causes of a system's incorrect behavior by means of observable symptoms
- **Prediction**. To infer the probable consequences of a given situation or of any change in the situation.
- **Design**. To make a configuration of a system under certain constraints along the basis of a set of possible alternatives.
- **Planning**. To design plans; that is, sets of actions.
- **Repairing**. To execute plans for administering foreseen remedies.
- **Control**. To monitor, diagnose, foresee and repair a system's behavior.

Now we should ask ourselves which of these application types mentioned above fit our system. As it is impossible to analyze all of the cases, we will consider a very frequent situation in industry in which the home system monitors

the functioning and triggers a warning when it detects an error. A final hypothetical expert system could serve as a support for the complete control, including the diagnosis of the causes of incorrect functioning, the determination of actions to be applied as remedies (prescriptions), its application and the follow-up of its effects. Logically, we should begin little by little, and an initial expert system would only be applied to those aspects close to monitoring; in other words,

the determination of a hypothesis for identifying the original causes that have given rise to a system's incorrect behavior which has been observed by warnings.

A system defined in this way must clearly be classified as a diagnostic- or interpretation-type expert system. This type of systems is very well known, they have been studied thoroughly and they were among the first to be recognized as successful: The MYCIN expert system [25], for example, diagnoses an infectious disease working from the set of symptoms that the patient is suffering from, his clinical history and the results of clinical analyses. Diagnosis and interpretation systems have been applied in very diverse fields other than medicine. Such is the case of PROSPECTOR [4], a consultant system that diagnoses the type of mineral bed found in a particular area using geological information observable on the surface. In the general case which concerns us, the symptoms would be the warnings and it would have to diagnose what was the cause or "disease" that brought it on.

a) Requirements Relative to the Reasoning Mode

Diagnostic-type applications give rise to a requirement of critical importance to the development shell, as the studies of a number of experts have corroborated [1, 3, 6, 11, 26]. This requirement is as follows:

*"A diagnostic system must use **backward chaining** as the control strategy of its inference engine."*

This is a highly important technical question that conditions the selection of the appropriate shell. In backward chaining, the inference engine begins working from one or several objectives or hypotheses and tries to justify them by means of rules that match these hypotheses. These rules will generate, in turn, new subhypotheses until the available data permit us to establish the certainty of successive subhypotheses and, therefore, the certainty of some of the initial

hypotheses. These systems are said to be objective-driven. In this way, in the case of medical diagnosis, symptoms give rise to hypotheses that are relative to the patient's illness. In order to corroborate these hypotheses, new subhypotheses are generated so as to provide the necessary clinical data, which give rise to analyses, tests, etc. This is the strategy used by practically all of the classic diagnostic/prescription systems [6]: MYCIN (EMCYN, the shell which descended from it), M.1 (Teknowledge), Personal Consultant (Texas Instruments), KES (Software A&E).

In forward chaining, all rules try to be successively triggered by matching all of the data the system knows to the antecedents of each rule. Triggered rules establish new data or facts. The process continues until no new rules can be triggered. The system is said to be data-driven. This is typically used in expert systems for design/planning-type applications, in which a constructive problem-solving technique must be applied. The solution is obtained by successively adding elements (actions, parts, ...) until obtaining a design that complies with the imposed restrictions. The classic example is the OPS5 shell used to develop the R1, XCON, XSEL, and PTRANS systems in order to design the configuration of DEC computer systems.

The control of forward-chaining systems is more complex than that of backward chaining: the former will habitually need rule-priority assignment techniques and the partition of rules into sets, which requires a greater amount of experience on the knowledge engineer's part. To be efficient, a forward system requires the use of sophisticated control techniques (Rete type). As the implementation of these techniques is complex, it is much easier to construct an efficient shell with backward chaining that with forward chaining [11]. Proof of this statement is the number of backward-chaining shells available on small computers with limited memory and CPU, as opposed to the scarcity of forward-chaining shells for these same platforms.

Vendors often affirm that their shells use a hybrid bi-directional chaining with forward as well as backward chaining. In other cases, the user is offered the possibility of selecting one mode or the other, even using two different inference engines. A tool that gives a similar quality for both types of control strategies would be desirable. Experience obtained from the evaluation of concrete systems shows that this is seldom possible in practice. In this way, for example, the comparative analysis carried out by Dr. W. Mettrey of Bell-Northern Research

[11] concludes by affirming that *"decisions that must be made during the initial design of a tool frequently result in strong support of one control strategy at the expense of the other".*

b) Fact Base Unfeasibility

In addition to the reasons noted in favor of backward-chaining motivated by application type, there exist others that are purely technical. In the usual expert system, the data or facts that make the rules trigger forward are found in the Fact Base. If this base is large, control and efficiency problems arise due to the fact that *all* of these data must be compared with the rule's antecedents; this situation is improved by the use of sophisticated algorithms (like Rete).

Habitually, it will not be possible to use a conventional fact base since there exists a great deal of data to be found in the home system's databases. A forward reasoning driven by such an amount of data with no control would inevitably make the system collapse. Selecting some data and temporarily transferring them to the fact base could be considered, but the selection criterion is not at all clear and would condition a priori the solutions that could be found.

In the case of backward chaining, the solution is immediate as we only need to compare the necessary data in order to validate successive hypotheses and subhypotheses. To achieve this, instead of asking the user questions —as MYCIN or PROSPECTOR did— the database is asked directly. The philosophy behind this would be that of a classic diagnostic system substituting the question generator in natural language for a direct query generator for the databases in their specific query language (thus, the importance of this requirement for the shell).

3.3 Rapid Prototype Development

Among the capabilities linked to shell development interface, rapid prototype development was mentioned. This capability requires a more detailed study. The development life cycle of an expert system is based on the incremental development technique with a three-stage repetitive cycle: 1-Design adjustments, 2-Implementation, 3-Verification and Validation.

The justification for this methodology is to be able to assess the following aspects in an early stage of the project:

- To evaluate the decisions made in the design stage which are relative to: problem conceptualization, knowledge representation, reasoning mode, etc.
- To evaluate the scope and validity of knowledge extracted from the expert system as well as the magnitude and complexity of the project.
- To serve as a demonstration in order to persuade the firm's management in favor of the project.

So that the prototype fulfills these objectives, it should be a small system yet it should possess the final system's main characteristics and should allow for the identification of aspects not considered in the initial design in order to redesign and re-implement. To achieve this, selecting a flexible and powerful shell which allows for the creation of a rapid prototype that works is a critical requirement.

4 Common Expert System Shells

In this section we give details about 31 common expert system shells. The comments about these shells are focused mainly on the critical aspects that have been considered above for the kind of applications this paper is concerned with. In general, the data about the shells analyzed in this section correspond to the versions that were available in the first trimester of 1995, some later versions may include further improvements or changes that are not included below. When the shells do not satisfy some of the critical aspects, these are only briefly mentioned; otherwise, a longer description of their capabilities is included. This by no means implies that the first shells would not be of great interest for applications other that those considered in this paper. The information summarized below is been taken from three main sources: in some cases from the data provided by vendors, in other it is based on recently published analysis, and also from the evaluation by the authors of this paper themselves. The available information may substantially vary for different shells.

- **EXSYS** (EXSYS, Inc.). User interface is not powerful, limited base language, production rules are not flexible and with limited capabilities, rule priority assignment is not flexible.
- **KEE** (IntelliCorp). Available on Unix platforms only under X-Windows. Base language is Lisp.

- **Kappa and ProKappa** (IntelliCorp). Available only for Sun and HP Unix platforms.
- **ACQUIRE and ACQUIRE-SDK**.(Acquired Intelligence Inc.). The vendor lacks a previous trajectory in AI products. The inference engine seems oriented towards forward chaining but it does not include the Rete algorithm. The cost is reasonable for PC ($995). It includes a software development kit for application integration, but information about the base language is lacking. User interface seems adequate.
- **RTworks** (Talarian). Only available for Unix and VMS platforms. 16Mb of RAM are recommended. Oriented towards applications that require real-time data acquisition with client/server architectures.
- **ART-IM and ART*Enterprise** (Inference Corporation). These shells have been superseded by other products (Eclipse and Rete++) developed by their own creators who left Inference Corporation to establish their own company. Moreover, they only allow for forward chaining, do not offer database access and are slower and more expensive that their competence.
- **KnowledgeWorks** (Harlequin). Since it is based on CLOS (Common Lisp Object System), it is only available for Unix workstations.
- **C-PRS** (ACS Technologies). This is not an expert system shell based on production rules, but rather it is based on *procedural reasoning* [8]. It is thought for applications requiring real-time response and it is not available for PCs.
- **GBB** (Blackboard Technology). This is an expensive product ($6500) for blackboard-type systems combining technologies such as expert systems, traditional procedures, neural networks, etc. It goes beyond the needs of an average expert system application. Its use requires experience in AI.
- **G2** (Gensym). This is a very expensive shell: two developer versions exist: —off line and on-line— with prices of $19,559 and $31,050 for PC. This cost is justified by their high performance qualities in terms of: real-time execution for on-line applications, a data sever that manages communications that are simultaneous to concurrent reasoning, executions with no stops, distributed systems integration, and a multiserver client/server architecture. It goes beyond the needs of an average expert system application.

- **ILOG RULES** (ILOG). Extension of OPS5 written in C++. Like its predecessor it only incorporates forward chaining. The information provided by the vendor does not permit a minimal assessment of this shell's capabilities.
- **KOOL 4x4** (Bull). This shell is based on objects and can be compiled as a C language module. Each rule is declared to be used with forward, backward or mixed chaining. It only runs on platforms of the DPX/20 family. The developer version costs $15,000.
- **Personal Consultant Plus** (Texas Instruments). This tool is oriented towards backward chaining and it is written in a dialect of Lisp. It is intended to serve as a quick introduction into expert systems on PC. Its weakest point is that it does not offer capabilities for embeddability or connection to databases.
- **Smart Elements and NEXPERT OBJECT** (Neuron Data). NEXPERT OBJECT is a very popular shell based on rules and objects. It incorporates forward and backward chaining integrated in the same rule format, together with automatic goal generation (opportunistic reasoning). This shell is now commercialized within a product called Smart Elements, which is an environment for intelligent applications development with a client/server architecture. Its architecture is based on elements, including: a Graphic User Interface (as part of **Open Interface Elements**), an element for data access and NEXPERT OBJECT. These elements can be integrated in a simple and portable fashion. This product offers complete portability over 35 hardware platforms, extendibility, as well as database and spreadsheet connectivity in a transparent way by means of built-in *bridges* (dBase III Plus, Excel, Lotus) or external bridges (Oracle, Ingres, Informix, DB2, SQL). Smart Elements supports and incorporates API´s for most current tools, the generated applications can be easily integrated within other type of applications or even third-party tools or libraries. The kernel of NEXPERT can be accessed as a C library. It allows for the implementation of friendly user interfaces in any native window system, this fact can be improved by the use of the module called Open Interface Elements. In addition, the product called **C/S Elements** is a powerful tool for portable client/server applications. As a whole, the system offers a great extendibility and scalability. The cost of Smart Elements is $1,900 for PC (for universities).

- **M.4** version 3.0 (Teknowledge Corporation). M.4 is a descendant of M.1, which appeared in 1984 and has been successfully applied in hundreds of expert systems. It was based on EMYCIN and conceived as a small but powerful tool on a PC. M.4 is written in C and adds new functionalities while still being easy to use and learn. It is based on backward chaining, incorporating procedural control and objects. Its strongest point is its flexibility and capacity to be embedded in other applications [13, 24] by means of its *Kernel Library* —which can be used as an executable file— or by DLL or VBX. All this is described in a 400-page *Embedder's Reference Guide*. It includes a DDE server for communications under Windows, as well as other interfaces for Visual Basic, Visual C++ and TTY. It also incorporates a connection with dBase III which is transparent for the knowledge engineer. In addition, it offers very convenient user interfaces, certainty factors, etc. There only exists a version for PC under DOS or Windows. **M.4 VB** is a specific product for integrating M.4 within Visual Basic with the possibility of accessing databases with ODBC.
- **Eclipse, The Easy Reasoner and VBXpert** (The Haley Enterprise). Eclipse was developed by the creators of the well-known ART shell, but it is implemented in C language (ART was written in Lisp: it was expensive and neither portable nor embeddable). It is based on the Rete algorithm but —like ART— in addition to forward chaining it supports *opportunistic* backward chaining. The main drawback is that this technique requires an experienced knowledge engineer. Eclipse is smaller and faster than CLIPS or ART, it offers automatic integration with databases, a powerful development environment and support for windows by means of DLLs. Its price is $999 for PC. The Easy Reasoner ($499) and VBXpert ($99) offer extensions for case-based reasoning and its use with Visual Basic.
- **Rete++** (The Haley Enterprise). This is an extended version of Eclipse based on C++. Therefore, it adds objects to Eclipse's functionalities. Its price for PC is $1,999.
- **Flex** (Logic Programming Associates). This shell is oriented towards forward chaining, though it supports especial rules for backward chaining. It uses frames and inheritance. Its base language seems to be Prolog, this fact limits its embeddability. According to the vendor's information, C procedures can be accessed from Prolog, it supports DLL and DDE under

Windows, as well as interfaces with databases: DBase III, Q+E library or others via ODBC.

- **GoldWorks III** (Gold Hill Inc.). This product is integrated in Windows 3. The company was one of the first to developed —some ten years ago— an implementation of Lisp (GCLISP) for PC. Good GUIs are offered as well as developer interfaces, tutorials, etc. It supports frames and multiple inheritance too. The publicity states that it works forwards, backwards or in a bi-directional mode. It includes and explanation facility and certainty factors. Though it is built on GCLisp, the vendor states that —through DDE— data can be transferred to or received from other applications, such as spreadsheets, databases, etc., as well as external calls to C functions. Its main drawback can be its lack of embeddability, since its base language is Lisp. The prices of this company's products are usually competitive.
- **OPS/83** version 4.0 (Production Systems Technologies). This is a successor of the well-known system OPS5. It is written in C and incorporates a combination of C, traditional procedural code, and improved control strategies. It uses ReteII algorithm —an improved version of Rete— resulting in a speed 5 to 7 times better than CLIPS. Previous versions were based on forward chaining, while this version introduces the so-called *generalized forward chaining*, which —according to its creators— "integrates the capabilities of forward chaining with the essential characteristics of backward chaining, but it is more flexible and easier to use than both of them". Its cost for PC is $1,950.
- **RAL** (Production Systems Technologies). It is defined as a *third generation system* which, due to technical improvements, it allows for a seamless integration of rules and objects in programs written in C language in such a way that it may surpass other shells such as CLIPS. It is thought of as a syntactic extension of C language. It also uses ReteII algorithm and *generalized forward chaining.* Its cost for PC is $1,950.
- **CLIPS** (Public domain). This shell was developed by NASA as an internal training tool for PC. They cloned the syntax and functionality of ART implementing it in C language. Since 1986 it is public domain. It has become rather popular in the last years, mainly due to its access at no cost. However, it presents serious drawbacks when compared with some of the above-mentioned commercial products: it is not a professional system, it has

been superseded by other commercial shells, and it only incorporates forward chaining. It does not offer good interfaces, vendor's support or help facilities. As its base language is C, it can be embedded in other C systems, but the developer must do it from scratch in C code: no interfaces to other applications or databases are offered. Other drawbacks are: rules in CLIPS cannot deal with arbitrary data types in C, CLIPS does not use C conventions for passing arguments to functions or returning values from functions. Trying to emulate backward chaining in CLIPS is very complex and without good results [11]. Eclipse or ART-IM —just to mention two— are clearly better that CLIPS. Its main interest is its use as a first introduction into expert system based on forward chaining rules.

- **ES Expert System, MIKE and RT-Expert** (Public domain). These are public domain shells. In general, public domain products are the result of experimental projects in universities or obsolete versions of commercial products. They may be technically correct, but they obviously lack the facilities of a commercial product: adequate graphic interfaces, documentation, edition and debugging tools, embeddability, external connections, etc. They can serve as a first contact with expert systems technology, but with the risk that this first impression may be negative. No vendor support can be expected. ES Expert System can be recommended as a useful shell for getting acquainted with the problems involved in the development of a knowledge base, it allows the user to practice the art of knowledge-base design. MIKE was developed by The Open University and is an adequate tool for instruction purposes, it comes with a reasonably well-developed environment and rather complete documentation. Real-Time Intelligent Systems Corporation offers an almost-free version of the their product RT-Expert ($64). There also exists a professional version. It is aimed at applications with strict constraints in terms of response times. It is conceived to be embedded in other applications, since the rules are finally converted into C modules.

5 Conclusions

In concluding this paper, we can affirm that the current engineering necessities call for expert systems embedded in the firm’s other systems, with the

possibility of access to the databases in an efficient way, with a shell that permits a rapid prototype development, with a friendly graphic interface whose software connects with other applications, and with clear advantages for an inference mechanism based on backward chaining.

Acknowledgments

The authors thank the students Soledad García Valls and Gabriel Recatalá for evaluating the ES Expert System, MIKE and RT-Expert shells.

Key words. Evaluation & selection, AI tools, implementation & integration strategies, knowledge-based sytems, expert systems.

References

1. Barry, R., "Expert Systems in Prolog", *PC AI*, Summer 1987, pp. 23-26.

2. Cervera, E., del Pobil, A.P., "A Hybrid Qualitative-Connectionist Approach to Robotic Spatial Planning", *Workshop on Spatial and Temporal Reasoning*, *International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

3. Clancey, W.J., "Heuristic Clasification", *Artificial Intelligence*, Vol. 27, pp. 289-350, 1985.

4. Duda, R.O. et al. "Development of the PROSPECTOR Consultant System for Mineral Exploration. Final Report". Artificial Intelligence Center, Stanford Research Institute, Menlo park, California, 1978.

5. Gevarter, W.B., "The Nature and Evaluation of Commercial Expert System Building Tools", *IEEE Computer*, Vol. 20, No. 5, pp. 24-41, 1987.

6. Harmon, P., King, D., *Expert Systems: Artificial Intelligence in Business*, New York, Wiley, 1985.

7. Hayes-Roth, F., Waterman, D.A., Lenat, D.B., *Building Expert Systems, Massachusetts*, Addison-Wesley, 1983.

8. Ingrand, F.F., Georgeff, M.P., "An Architecture for Real-Time Reasoning and System Control", *IEEE Expert*, Vol. 7, No. 6, pp. 33-44, 1992.

9. Marcos, M., Moisan, S., del Pobil, A.P., "Verification and Validation of Knowledge-Based Program Supervision Systems", *IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, Canada,1995 (enviado).

10. Martorelli, W.P., "PC-Based Expert Systems Arrive", *Datamation*, Vol. 34, No. 7, pp. 56-66, 1988.

11. Mettrey, W., "A Comparative Evaluation of Expert System Tools", *IEEE Computer*, February 1991.

12. Mettrey, W., "An Assessment of Tools for Building Large Knowledge-Based Systems", AI Magazine, Vol. 4, No. 8, pp. 81-95, 1987.

13. Murphy, T., "Wil you loan me your M.4", *AI Expert*, 1993.

15. del Pobil, A.P., Muñoz, C., García, J.L., Bret, A., "Implementation du Système Expert Iroise dans un contexte multitâche en mode serveur sur réseau ethernet", *A.M.A.I.A.* (Architectures, Méthodes, et Applicationes en Informatique Avancée), Bayonne (Francia), 41 págs., 1988.

19. del Pobil, A.P., “Selection of Experst System Shells”, Tech. Report, Dept. of Informatics, Universitat Jaume-I, 1995, (in Spanish).

20. del Pobil, A.P., Serna, M.A., "Robot Motion Planning", in *Applications of Artificial Intelligence in Engineering VIII*, edited by G. Rzevski, J. Pastor and R.A. Adey, Elsevier Applied Science, London, pp. 515-537, 1993.

21. del Pobil, A.P., Serna, M.A., "Solving the Find-Path Problem by a Simple Object Model", *Proc. 10th European Conference on Artificial Intelligence (ECAI-92)*, Vienna, Austria, pp. 656-660, 1992.

22. del Pobil, A.P., Serna, M.A., *Spatial Representation and Motion Planning*, Springer-Verlag, 1995.

23. Rothenberg, J. et al., *Evaluating Expert Systems Tools: A Framework and Methodology*, RAND Corporation, California, 1987.

24. Schmuller, J., "M.4: Something for Everyone", *PC AI*, Nov./Dec. 1993.

25. Shortliffe, E.H., *Computer-Based Medical Consultation*, New York, American Elsevier, 1976.

26. Stylianou, A.C., Madey, G.R., Smith, R.D., "Selection Criteria for Expert System Shells: A Socio-Technical Framework", *Communications of the ACM*, Vol. 35, No. 10, pp. 30-48, 1992.

27. Stylianou, A.C., Smith, R.D., Madey, G.R., "An Empirical Model for the Evaluation and Selection of Expert System Shells", *Expert Systems with Applications*, Vol. 8, No. 1, pp. 143-155, 1995.

28. Stylianou, A.C., Smith, R.D., Madey, G.R., Expert System Application Types: An Empirical Classification", Technical Report, The Belk College of Business Administration, University of North Carolina at Charlotte, 1993.