CHAPTER 6

Selection of an appropriate tool for ES development

6.1. Introduction

Development of expert system applications is experimental, expensive and difficult. A number of factors are to be considered for the successful development of an expert system. Selection of an appropriate expert system problem domain, knowledge acquisition and formal representation, system construction, validation and testing are typically important. The degree of complexity of the above tasks are fairly high. Moreover it is fairly easy to mismanage the situations in different steps. Different expert system tools have been developed to improve the situation. These tools provide the basic components needed to construct an expert system. Because the choice of a development tool can determine the success of the resulting expert system application, the characteristics of these tools and of the target problem must be clearly understood before a selection is made. More specifically, these tools are used to reduce the burden on a developer and in a more cost-effective way. Such tools enable developers to implement expert system applications by entering knowledge specific to the particular problem they are attempting to solve. Although, there are tools to assist in different phases of such development, we are confined here to the matter of selection of tools during actual system construction process after the selection of a problem domain, and the knowledge acquisition and representation methods. So the major advantages that arise from using a tool can be summarized as follows [1].

a) Use of tool can improve the overall quality and reliability of the resulting expert system.

b) Tolls relieve the expert system builder from having to deal with low level programming.

c) Tolls allow the expert system builder to focus on the modeling of the expert system domain.

d) Tools offer facilities for the acquisition and modification of the expert system’s knowledge.

A large number of expert system building tools have been introduced since late 1970s. These tools range from high level programming language to intelligent
editors to complete shell environment systems. A number of commercial/freeware products are now available (table 6.1), some are capable of running on medium size PCs while other require large systems such as LISP machines, minis, or even main frames.

Table 6.1. Expert systems tools [2].

<table>
<thead>
<tr>
<th>Freeware tools</th>
<th>Description: This is a modular, configurable, hybrid environment for developing expert systems. It provides the following knowledge representation formalisms: frames, rules, logic (Prolog) and constraints. It requires Common Lisp. Platforms: Mac, and UNIX.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BABYLON</td>
<td>Description: The ES Expert system development tool supports backward/forward chaining, and fuzzy set relations. Platforms: PC.</td>
</tr>
<tr>
<td>ES</td>
<td>Description: This shell can be used in a variety of problem domains and supports backward and forward chaining. Its knowledge representation schemes include frames, rules and procedures. Support is also present for fuzzy logic and certainty factor maintenance. It includes a blackboard architecture. The user interface utilizes the Symbolics windowing system and is menu and mouse driven. Platforms: Symbolics Lisp Machines, Genera 7.2.</td>
</tr>
<tr>
<td>GEST (Generic Expert System Tool)</td>
<td>Description: A forward-chaining rule-based tool written in C by NASA. It can be easily embedded in other applications and includes an object-oriented language called COOL. Platforms: DOS, Windows, VMS, Mac, and UNIX.</td>
</tr>
<tr>
<td>CLIPS (C Language Integrated Production System)</td>
<td>Description: A number of tools are available to be linked with CLIPS. DYNACLIPS is a set of blackboard, dynamic knowledge exchange, and agent tools implemented as a set of libraries that can be linked with CLIPS. Platforms: DOS, Windows, VMS, Mac, and UNIX.</td>
</tr>
<tr>
<td>DYNACLIPS (DYNaamic CLIPS Utilities)</td>
<td>Description: This version of CLIPS provides handling of fuzzy concepts and reasoning, in addition to the other CLIPS features. Platforms: DOS, Windows, VMS, Mac, and UNIX.</td>
</tr>
<tr>
<td>FuzzyCLIPS</td>
<td>Description: A rule-based system with allows for integration of the expert system with C or C++ code. Platforms: DOS.</td>
</tr>
<tr>
<td>RT-Expert for DOS, Personal Edition</td>
<td>Description: It supports forward and backward chaining, an object oriented knowledge representation, graphics, and calls to/from other languages (C, Pascal). Platforms: DOS, OS/2, SunOS, Microsoft Windows, and VMS.</td>
</tr>
<tr>
<td>Aion Development System (ADS)</td>
<td>Description: Machine-learning software, an add on to XpertRule, that uses genetic algorithms to optimize solutions. Platforms:</td>
</tr>
<tr>
<td>Analyser</td>
<td>Description: An integrated C++ based development tool for building expert systems. Its graphical development environment supports rule based and case based reasoning (CBR), OOP, DBMS integration and GUI creation.</td>
</tr>
<tr>
<td>Platform/Development Tool</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Doctus KBS</td>
<td>Knowledge-Based Expert System Shell 'Doctus' uses deduction also called Rule-Based Reasoning and induction, which is the symbolic version of Case-Based Reasoning, enhanced with reduction. The Knowledge Import component of the system is designed to retrieve both soft and hard information from external sources, which makes it appropriate for data mining. Doctus is also equipped to export knowledge in various forms of intelligent agents via its Knowledge Export module. Platforms: MS Windows.</td>
</tr>
<tr>
<td>EXSYS Professional</td>
<td>An easy to learn rule-based expert system shell and an excellent educational tool which comes with many examples and a good tutorial on developing expert systems. It features backward and forward chaining, blackboarding, fuzzy logic, and frames. SQL interface and linking to database and spreadsheet programs such as Lotus 1-2-3 is supported. Platforms: DOS, Windows, Macintosh, UNIX, and VAX.</td>
</tr>
<tr>
<td>EXSYS RuleBook</td>
<td>Development tool that allows the building of expert systems using tree diagrams. Platforms: Windows, Macintosh.</td>
</tr>
<tr>
<td>EXSYS Linkable Object Modules</td>
<td>Allows customization of EXSYS programs, addition of up to 100 user-defined C functions, embed neural networks, and add DDE links to other programs. Platforms: DOS, Windows, Macintosh.</td>
</tr>
<tr>
<td>KEE (Knowledge Engineering Environment)</td>
<td>KEE supports a variety of knowledge representation schemes including object-oriented frame language. The inference engine supports both forward and backward chaining. It allows for linking to several databases. Its interactive graphics interface is one of the most sophisticated available among expert system tools. Platforms: PC, VAX, Sun.</td>
</tr>
<tr>
<td>M.4</td>
<td>An expert system development tool that includes support for rule-based procedural control and object-oriented representation. Provides interface to Visual Basic and Visual C++ and supports forward and backward chaining and DDE and DLL support. Platforms: DOS, Windows, Sun, and Mac.</td>
</tr>
<tr>
<td>Nexpert Object</td>
<td>An expert system development tool with a graphical user interface. It features a rule-based and object-based inference engine. It allows for the interfacing with databases, programming languages and other applications. Platforms: DOS, Mac, UNIX, and VMS.</td>
</tr>
<tr>
<td>OPS83</td>
<td>OPS 83 is a rule based system that is a successor of OPS5. It is written in C and allows for the integration of applications written in C. OPS 83 supports generalized Forward Chaining, a control structure which allows rules to be more expressive. Platforms: DOS, OS/2, VMS, and UNIX.</td>
</tr>
<tr>
<td>RT-Expert</td>
<td>A rule-based system with allows for integration of the expert system with C or C++ code. Platforms: UNIX, DOS, Windows, and VMS.</td>
</tr>
<tr>
<td>XpertRule</td>
<td>A windows-based expert system development tool which utilized genetic algorithms for optimization. It generates code in C, Pascal and COBOL. Platforms: MS Windows/PC.</td>
</tr>
</tbody>
</table>
In the next section, we discuss some issues in selecting a tool for implementation during an expert system development. The capabilities of ES-building tools are discussed in section 6.3. In section 6.4, we have analysed the features and capabilities of an object-oriented hybrid ES tool Level5 Object, at our disposal. Section 6.5 contains the requirements vs. the capabilities of the selected tool. Lastly, we provide some discussion.

6.2. Points to contemplate

This section presents some potential inconveniences faced by an expert system designer and discusses the classification of tools.

6.2.1. General purpose tool not available

There would have been no much headache of such selection if we have been provided a general purpose tool for our use. Unfortunately, there is no such general purpose tool simply because human beings commonly utilise a knowledge based approach rather than a general purpose approach for problem solving [3]. As a result, a range of tools are provided applicable in a wide variety of domains. One has to select his/her suitable tool for his/her domain.

6.2.2. Single or multiple tools?

Due to the evolutionary nature of an expert system development, one may find it worthwhile to use one tool for prototype system and the other for target system. Obviously, this adds one confusion - which type(s) of tools would be suitable for prototype development and which would be suitable for target system development. It is rather difficult to offer sound guidelines here. Two typical approaches may be useful: (i) solving the chicken-egg problem [4], and (ii) use one of the large, hybrid object-oriented toolkits [5] throughout the total developmental phases. In the first approach, it is assumed that few expert systems development efforts are well formulated in advance. The problem domain is complex, domain knowledge and expertise are ambiguous and the nature of the problem is such that complete unfolding is only possible after considerable exploration and experimentation. This is where one has to address the chicken-egg problem. One can choose an appropriate tool only after understanding the total requirements the problem domain lays on an expert system; but this may only be possible after experimentation on building a prototype which requires choosing a tool. This is where two types of tools may be considered useful. During demonstration/research prototypes, one should use a tool having fast prototyping property e.g. Prolog [6].
The second approach assumes that selection of such appropriate implementation tool should only be done after complete requirement analysis of the problem domain lays on an expert system. In most of the situations this may prevail. Even though the total requirement analysis is not complete, it might be advantageous to use a comprehensive tool assuming that the simpler rule-based production systems do not qualify; model-based reasoning would be more appropriate. Potential difficulties in using multiple tools (discussed below) can be reduced here substantially. The objectives of demonstration/research prototypes can be fulfilled using the simple structures of the tool. More complex structures of the tool can be used for production system.

Use of multiple tools may lead to the following problems: (i) Financial investment; may not be practical in many cases, (ii) Man-years investment; before using any such tool, one has to be conversant with the intricacies of the tool. Moreover, once domain experts and knowledge engineers have been trained on a particular system, it is not simply feasible to be retrained on another potentially suitable system; (iii) Interfacing problem; usually the architectural design and functionality of each tool is different, one has to interface such two or more tools. This would require an extra degree of expertise involving a fair amount of man-years. The original objectives of using tools e.g. less time, less effort are thereby get challenged.

As a worst case, no such existing higher level tools might be adequate to satisfy the requirements of the problem domain. This is where one has to design his/her own architecture. Here, one has to fix up his/her mind whether the goal of the system development is to develop a system for actual use or to make major advances in the state-of-the-art of expert system technology. As observed by Preran [7], it is not wise to attempt to achieve both of these goals simultaneously simply because it is really a formidable task. So, a system development for actual use may demand pruning of certain characteristics of the problem domain within the confines of current expert system technology.

6.2.3. In search of a bird after constructing a cage

Currently, a good number of higher level tools are available (e.g. shells or toolkits). Now one can observe an interesting phenomenon [8]. In most cases, peoples are in search of birds after the construction of a cage. That is, the members of a project team are forced to in search of a problem domain which suits the solution techniques of the tool already chosen. This approach, obviously, restricts the natural flow of selection of a problem domain of our society. Therefore, the reverse approach, i.e. selecting an expert system
problem domain and then go for selecting an appropriate tool for implementation, is much more natural.

6.2.4. Exaggerated claims from vendors/agents

Vendor literature, demonstrations and reference manuals are subject to exaggerated claims [9]. It is rather very difficult for the end users to distinguish between facts and hyperbole. It may not always be possible to go through the detailed experimental verification of these claims before the actual procurement. Even, sometimes, agents do suppress the potential demerits of the tools for selling his/her product.

6.2.5. Non-standard terminologies

For a better comparison of tools, standard terminologies with their standard definitions and actions are really useful and thereby ease the process of selection of such tools. But, unfortunately, some tools do not agree in terminologies. For example, in KEE, frames are called units, properties of units are called slots, and properties of slots are called facets. But, in S1, frames are called classes, properties of classes are called attributes, and properties of attributes are called slots. Similarly, the term rule used in ROSIE, ART and RULE MASTER are different in performing actions. So, the terminologies with non-standard definitions and actions add an extra degree of difficulty, the prospective users face when selecting tools.

6.2.6. Miscellaneous issues: Price, training and documentation support

‘High price, good quality - more facility’ is, however, considered as an indicator of a product. But, from functionality and performance point of view, price is not necessarily an indicator of suitability. A tool costing less may be more suitable at per the requirements of the problem domain at hand than a high cost tool. For the ease of use and for the quick exploitation of the potential features of the tool, a comprehensive training should be considered mandatory.

6.2.7. Language, shell or toolkit

An appropriate tool selection should also be guided by the relative merits and demerits of these three classes of tools. In general terms, the shells provide the upper level of a stratum of tools, the lower level is being provided by the languages and the middle level is being provided by the toolkits.
Languages are applicable quite generally and virtually any type of expert system can be designed. Expert system can be constructed with one of many programming languages ranging from AI languages to standard procedural languages.

The symbolic manipulation languages provide an efficient way to present AI-type objects. The two major languages are LISP and PROLOG. With these languages, the programming and debugging procedures can frequently be done much faster.

LISP is one of the oldest general purpose languages. It is oriented towards symbolic computation and can conveniently manipulate symbols and their relationships. PROLOG is another popular AI language. Its basic idea is to express statements of logic as statements in programming language. This logic itself could be used directly as a programming language. PROLOG is based on subset of first order logic (predicate calculus), so one of the techniques of knowledge representation is first-order logic.

The components of ES like knowledge acquisition subsystem, inference engine, explanation facility interface subsystem and knowledge base management facility when aggregated, is called an ES shell. The knowledge base is the content of the shell. There is no need to program the subsystems of the shell for every application, all one has to do is insert the necessary knowledge. By using the shell approach, expert systems can be built much faster. Shells do have limitations and disadvantages. Shells are inflexible and it may be difficult to fit them to non-standard problems and tasks [10].

ES development systems and other building aids that support several different ways of representing knowledge and handling interfaces are frequently referred to as knowledge engineering tools or toolkits. They may use frames, object-oriented programming, semantic nets, rules and meta-rules, different types of chaining, monotonic reasoning, inheritance techniques and more. The toolkits are normally hybrid systems that permit a programming environment to build complex specific systems. Toolkits are more specialized than languages. They can increase the productivity of system builders. Although toolkits require more programming skills than shells, they are more flexible. Working with toolkits makes ES more economically justifiable, especially when they are being developed on personal computers [10].
The metrics like applicability, abstraction, facilities and costs of hardware, software and training may be considered useful in the comparison process [3] as discussed below.

- **Applicability**

  Languages are applicable quite generally and virtually any type of expert system can be designed. On the other hand, shells are rather specific in this context. The good matching of the requirements the domain lays on an expert system and the facilities the shell offers is the key to success of the development of an expert system. The toolkits should have the generality of the language approach but also contain specific representations and control strategies.

- **Abstraction**

  The level of abstraction is low in the language approach and medium in the shell approach. On the other hand, toolkits provide a rich set of abstraction.

- **Facilities**

  In the toolkit, the facilities are most rich. Shells offer medium facilities. In a language approach, we get limited facilities but, however, any facility missing in a language may be provided by programming.

- **Costs**

  (i) **Hardware**
  
  In the case of languages and shells, the costs of hardware are generally quite low compared to toolkits. This is simply because toolkits often demand specialised hardware whereas languages and shells run on PC or on workstations. Although, recent versions of some toolkits run on PC but the memory / backup size is reasonably high.

  (ii) **Software**
  
  The costs of languages and shells are more or less same. But, toolkits are normally more costly than other two.

  (iii) **Training**
  
  In general, the shells require a fairly less efforts and short time for learning. But, however, languages require more extensive efforts and training period.
The toolkit approach appears to be superior to other two approaches on the consideration of applicability, abstraction and facilities offered. But, however, it appears to be inferior to language and shell approaches on the consideration of costs of hardware, software and training. Moreover, although, a toolkit offers a good number of facilities, these may not fulfill all the requirements of the problem domain lays on an expert system. The programming facility, if any, of a toolkit is expected to fulfill such requirements. Once again, one has to be a master of a language like LISP or PROLOG which is provided by a toolkit.

At present there are a large number of toolkits available in the market. In such a situation, it may be pertinent to ask whether it is practically feasible or not for an end user to turn each and every one. This should not be an impossible task but this may lead to an unacceptable delay in achieving the ultimate objective of selecting such a tool.

6.2.8. Left no stone unturned - Is it practically feasible?

Bundy [11] provides a catalog of over 250 software products and AI techniques. Hopefully, this number should be over 1000 at the end of 2009. In such a situation, it may be pertinent to ask whether it is practically feasible or not for an end user to turn each and every stone. This should not be an impossible task but this may lead to an unacceptable delay in achieving the ultimate objective of selecting such a tool. Obviously, this demands a fast pruning mechanism.

6.2.9. Potentially active research field

It may not be possible to select the best one but a better one due to the evolutionary nature of this potentially active research field. One may find a better tool tomorrow satisfying more need of the problem domain. But, to develop a system for practical use one has to be confined to the present ES-technology where all the requirements may not be satisfied.

6.2.10. Any unique framework?

Because of the so many turbulent features just creating the instability in the selection process, the tool evaluation and selection can not entirely be mechanical. Here, human expertise and judgment will certainly play a significant role especially for the pruning process. But, obviously, this might lead to different solutions of the same problem which again demands a more formal mechanism. To what extent this formalisation would be achieved? Answering this question and suggesting a formal method are really formidable
It is, rather little bit easier to suggest a general framework for the problem. Rothenberg [4] suggests a framework with eight fold steps emphasising ‘matching a tool to its intended use’ rather than simply ‘matching a tool to a problem’. This framework might be worthwhile in many cases but it involves a larger number of criteria, may not lead to a manageable situation.

6.3. ES building tools

An expert system tool, or shell, is a software development environment containing the basic components of expert systems. Associated with a shell is a prescribed method for building applications by configuring and instantiating these components. Some of the generic components of a shell are shown in Figure 6.1 and described below. The core components of expert systems are the knowledge base and the reasoning engine.

![Figure 6.1. Basic components of Expert System tools](image)

a) **Knowledge base**: A store of factual and heuristic knowledge. An ES tool provides one or more knowledge representation schemes for expressing knowledge about the application domain. Some tools use both frames (objects) and IF-THEN rules. In PROLOG the knowledge is represented as logical statements.

b) **Reasoning engine**: Inference mechanisms for manipulating the symbolic information and knowledge in the knowledge base to form a line of reasoning in solving a problem. The inference mechanism can range from simple *modus ponens* backward chaining of IF-THEN rules to case-based reasoning.

c) **Knowledge acquisition subsystem**: A subsystem to help experts build knowledge bases. Collecting knowledge needed to solve problems and
build the knowledge base continues to be the biggest bottleneck in building expert systems.

d) **Explanation subsystem**: A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at to justifying the need for additional data.

e) **User interface**: The means of communication with the user. The user interface is generally not a part of the ES technology, and was not given much attention in the past. However, it is now widely accepted that the user interface can make a critical difference in the perceived utility of a system regardless of the system's performance.
6.3.1. Tools’ capabilities

Capabilities of a tool demands more attention, rather than the specific features the tool provides for achieving or supporting those capabilities. Highlighting capabilities means highlighting the functionalities of a tool rather the specific implementation of functionality. Some representative potential capabilities and the corresponding supporting features are identified here as presented in table 6.2 which includes the suggestions of Rothenberg [4].

Table 6.2. Capabilities of tools with supporting features.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Examples of Supporting Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic processing</td>
<td>Arithmetic operators, extended floating point</td>
</tr>
<tr>
<td>Built-in-functions</td>
<td>Mathematical, statistical, string, type conversion</td>
</tr>
<tr>
<td>Certainty handling</td>
<td>Certainty factors, fuzzy logic</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Distributed processing, parallel processing</td>
</tr>
<tr>
<td>Consistency checking</td>
<td>Knowledge base syntax checking</td>
</tr>
<tr>
<td>Data type handling</td>
<td>Numeric, string, time, simple, compound</td>
</tr>
<tr>
<td>Documenting development</td>
<td>Assumption/rationale history, code/data annotation</td>
</tr>
<tr>
<td>Explanation</td>
<td>Execution trace, knowledge base browsing</td>
</tr>
<tr>
<td>Inference &amp; control</td>
<td>Iteration, forward / backward chaining, inheritance</td>
</tr>
<tr>
<td>Integration</td>
<td>Calling other languages, inter-process calls</td>
</tr>
<tr>
<td>Internal access</td>
<td>Tool parameter setting functions, source code</td>
</tr>
<tr>
<td>Knowledge acquisition</td>
<td>Rule induction, model building aids</td>
</tr>
<tr>
<td>Knowledge-base editing</td>
<td>Structure editors, graphic rule lattice</td>
</tr>
<tr>
<td>Life Cycle</td>
<td>Tool support for target system life cycle support</td>
</tr>
<tr>
<td>Menus</td>
<td>Goals, Reasoning, tools</td>
</tr>
<tr>
<td>Meta-knowledge</td>
<td>Rules controlling inference, self-organizing data</td>
</tr>
<tr>
<td>Optimization</td>
<td>Intelligent look-ahead, caching, rule compilation</td>
</tr>
<tr>
<td>Presentation (I/O)</td>
<td>Text, graphics, Windows, forms, mouse, keyboard</td>
</tr>
<tr>
<td>Representation</td>
<td>Rules, frames, procedures, objects, simulation</td>
</tr>
</tbody>
</table>

Let us now consider a large-hybrid-object-oriented toolkit - Level5 Object; a potential tool at our disposal. Level5 Object has certain good features and capabilities to be selected as an implementation toolkit for present problem domain. Level5 Object offers facilities for data abstraction, message sending, object classification, built-in user friendly graphic interfaces, rules and rule-groups, bi-directional chaining and many more. Non-monotonic reasoning or truth maintenance is also supported by Level5 Object. But however, it is fair to state that it has rudimentary capability of handling inexactness. The features and capabilities of Level5 Object are discussed in the next section.
6.4. Level5 Object

6.4.1. What is Level5 Object?

Level5 Object is a software development tool kit from Information Builders, USA. It is the application development tool to combine client/server technology, object-oriented programming, graphical user interfaces, and knowledge based systems [12].

Level5 Object is a software development tool kit. Even if one has little programming experience, he / she can use it to create complex applications in an easy, consistent, and maintainable fashion.

Level5 Object is an environment. It contains all the tools necessary to solve a very wide range of problems. The Level5 Object toolbox contains an integrated set of editors that help one rapidly create software solutions. From rapid prototypes to large, mission-critical applications, Level5 Object is a proven winner as demanded by its developer.

Level5 Object is a development tool and a delivery vehicle. Once an application has been created using the Development System, it can be delivered to end-users with the Run-Only System. The Run-Only System provides a variety of delivery choices emphasizing smaller, faster systems. These systems can be encrypted to provide enhanced security.

Level5 Object uses a high-level language called PRL. PRL is designed to be simple to learn and read, and is similar to natural English. Although we rarely see it because of the interactive editors, it is there to provide maximum flexibility and accuracy when we develop an application. All of the elements of the software we wish to create can be expressed in the PRL language. The text of the application can be sent to other hardware platforms and operating systems, compiled, and executed. We can therefore create our software solutions where it is most convenient and inexpensively deliver them to other hardware platforms.

Level5 Object is well-connected. It contains built-in access to over 60 local and remote data-bases and servers, access to all the common local database formats and SQL servers, interfaces to external programs, communication paradigms, text files, timers, and custom interfacing options.
Level5 Object is an expert system. One can use it to create ‘smart’ systems. It can solve real world problems. Using simple rules that can reason or pattern matching triggers that can react to situations, Level5 Object can provide consistent, educated answers to the people who need it.

Level5 Object contains the following integrated array of powerful tools:

- True objects providing the efficiency of object-oriented programming.
- Graphical User Interface (GUI) development editors, forms and display builders, and control over all aspects of the user interface.
- Complex logic capabilities, business rules, triggers, agendas, procedural and non-procedural modules.
- Robust and seamless database access, SQL, object-oriented databases, and client / server architectures.
- Complete set of integrated debugging tools, stepping, breakpoints, traces, and reasoning.
- 100% portability to other hardware and operating system platforms.
- Compiled execution for efficient application speed and size.

6.4.2. What kinds of problems are best solved with Level5 Object?

Level5 Object has consistently shown itself to be an effective tool for solving certain classes of application problems. Here is a short list of the kinds of problems one can solve with Level5 Object. However, because it is a general purpose tool with a broad range of capabilities, one may not be limited to this list.

- Scheduling
- Resource and constraint management
- Regulation compliance and conformance
- Diagnostics
- Client/Server
6.4.3. Capabilities of Level5 Object

Let us now have a look on the capabilities of Level5 Object as follows:

- Arithmetic processing
- Built-in-functions
- Certainty handling
- Consistency checking
- Data type handling
- Documenting development
- Explanation
- Inference and control
- Integration
- Internal access
- Knowledge acquisition
- Knowledge base editing
- Life cycle
- Menus
- Meta-knowledge
- Optimization
- Presentation (I/O)
- Representation

6.5. Requirements vs. capabilities

Knowledge bases can be exported to a text file using PRL syntax in Level5 Object. PRL (Production Rule Language) is Level5 Object's application development language. One can see the underlying PRL structure of his / her application when one exports it to a text file. One can edit this file and also send it to Level5 Object running on other hardware platforms and operating systems. This process lets one create applications where it is most convenient and deliver them to the platforms one wants. But, however, when export of an application to be transported to character-based platforms, such as VAX / VMS or MVS, some elements will import, but will have no effect when the application is run. For example, picture-boxes do not appear in character-based displays. So, with the PRL structure one can achieve the portability.
From the PRL source text, one can have the facility of quick and easy modification of existing knowledge base. So, the proper patching work is not difficult here. With the different editors one can easily add or delete an object, its attributes, methods and the consistency checking is monitored by the Level5 Object itself.

Probably, one of the limitations of current tools is in their handling of inexactness of information. Level5 Object manages only one form of inexactness i.e. uncertainty, in the form of certainty factors (-2..100). MYCIN style of approach has been used here. However, it is not capable of handling other forms of inexactness as identified for our problem domain e.g. fuzzy information, simultaneous occurrence of uncertainty and fuzziness, and uncertain-fuzzy. But, however, implementing NMR may be more or less easy, although, it is very difficult to pinpoint the objects / attributes / rules / methods / demons affecting the absurd conclusion. But, once, they are identified it is easy to upgrade the information in knowledge base using different convenient editors.

Level5 Object provides a complete access to the inferential reasoning process while running a knowledge base. Its analytical reasoning facilities, such as season monitor, historical traces, single-stepping and breakpoint enable developers (and when necessary, end users) to view all the current status of the inference engine; examine and change the state of any facts in a knowledge base; review the answers provided to Level5 Object queries; and follow the line of reasoning being pursued. By activating a debug window while running an application, developers can observe and trace the reasoning process. In single-step mode, Level5 Object pauses after each event the inference engine processes, allowing the developer to view the action before resuming the session. So, a comprehensive explanation facility is being provided by Level5 Object.

Although Level5 Object manages objects and attributes from editor's panel which essentially freeze the knowledge base before running an application, but, however, it can manage instances of an object dynamically i.e. during running an application using MAKE and FORGET commands. The system can learn the situations of using MAKE and FORGET for the defined rules / methods / demons. The repetitive questionnaire of same kind during interrogation with a child and/or with parents / guardians can be avoided by suitably navigating the question-answer sequence. One can save recommendations of a typical session using an external database (typically dBASE III+). The required
structuredness and modularity are being assured by the object-oriented design strategy of Level5 Object.

Level5 Object supports the development of large applications through the use of knowledge-based subroutines that allow knowledge to be grouped modularly or chained to knowledge bases. Besides being easier to maintain, subroutines enable a host knowledge base to resume processing where it left off. Developers can navigate between modules and even redefine views of the knowledge domains.

6.6. Discussions

In this chapter we have tried to analyse the potential inconveniences we faced during the selection of the tool for the present work. We found Level5 Object: a suitable expert system development tool for the development of the present decision support systems presented in this work. It is important to consider capabilities of a tool in addition to the features of a tool highlighted by the vendor(s). However, in addition to Level5 Object, we have used web-enabled tools for the development of web-accessible systems in the present thesis.
References


