Chapter 5

Selection of an appropriate tool

5.1. Introduction

A number of factors are responsible for a 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. Not only the degree of complexity of these tasks are fairly high but also it is fairly easy to mismanage. Different expert system tools have been developed to improve the situation. More specifically, these tools are used to less the burden on a developer and in a more cost-effective way. 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.

A large number of expert system building tools have been introduced, both by the academic community and industry since the introduction of the first successful expert systems in the late 1970s. These tools range from high level programming language to intelligent editors to complete shell environment systems. A number of commercial products are now available (chapter 4, section 4.11), some are capable of running on medium size PCs while other require large systems such as LISP machines, minis, or even main frames.

In the next section, we discuss some points to ponder in selecting a tool for implementation during an expert system development. The capabilities of ES-building tools are discussed in section 5.3. In section 5.4, we have analysed the features and capabilities of an object-oriented hybrid ES tool Level5 Object, at our disposal. Section 5.5. contains the requirements vs. the capabilities of the selected tool. Lastly, our conclusions and discussion are summarised.

5.2. Points to ponder

This section presents some potential inconveniences faced by an expert system designer and discusses the classification of tools. This section also intends to highlight the inconveniences particularly faced by the author in the selection process in connection with the present work.

5.2.1. No general purpose tool

The headache of selecting an appropriate tool would have been relieved if we have been provided a general purpose tool for our use. Unfortunately, it is not possible to construct such a general purpose tool simply because human beings commonly utilise a knowledge based approach rather than a general purpose approach for problem solving [1]. As a result, a range of tools are provided applicable in a wide variety of domains. For a successful development of an expert system, it is essential that an appropriate tool is being selected.

5.2.2. Single or multiple tools

In a particular situation, a single tool may not be adequate to fulfil the requirements the problem domain lays on an expert system. 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 [2], and (ii) use one of the large, hybrid object-oriented toolkits [3] 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 [4]. 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. The amount of man-years would be increased simply by a factor of two or three depending on the number of tools used. 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 challenged.

As a worst situation, 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 Prerau [5], 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.

5.2.3. In search of a bird after constructing a cage

With the advent of increasing number of higher level tools (e.g. shells or toolkits) one can observe an interesting phenomenon [6]. 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. Obviously, this may lead to a situation of having no higher level tool suitable for the problem domain.

5.2.4. Exaggerated claims from vendors / agents

It is rather very difficult for the end users to distinguish between facts and hyperbole. Vendor literature, demonstrations and reference manuals are subject to exaggerated claims [7]. 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.

5.2.5. Non-standard terminologies

Standard terminologies with their standard definitions and actions are really useful for a better comparison of tools and thereby ease the process of selection of such tools. 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.

5.2.6. Miscellaneous issues : Price, training and documentation support

"High price, good quality - more facility" is, however, considered true. 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. The trainer should be an expert preferably a core member of the group responsible for the development of the tool. In different developed countries, the source of most advanced tools, this can easily be observed. Most of the vendors have their online facilities to support the customers; this facility can easily be enjoyed by the customers having advanced communication tools. For us, in India, the situation is not so favourable. Usually, the vendors sell their products through agents; the agents are not well-equipped with the properly trained experts. Even, the agents are, sometimes, reluctant to offer an on-site demonstration. Frequent on-line assistance is again a costly affair; may not be feasible for every customer. The situation may be improved with good documentation including a user's guide, a reference manual and an architecture manual and some demonstration examples with clarification.

5.2.7. Language, shell or toolkit

The selection of an appropriate tool should also be dictated by the relative merits and demerits of these three classes of tools. Generally speaking, 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. Some expert system development tools are written in non-AI languages, for example TIMM,

INSIGHT 2, EXSYS etc. Some specific expert systems have also been completely programmed in such languages. The most likely reason is that no other language is available for the hardware on which the expert system is to run. Languages like FORTRAN suffer from a disadvantage in that they can manipulate effectively only a small range of AI data types. In writing AI, the programmer may need to handle objects like rules, semantic nets and explanations. It is sometimes necessary to build and split rules or construct nets continuously which ultimately needs a garbage collection process to clean the memory filled with intermediate results.

The AI or 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. LISP allows programmers to represent objects like rules and nets as lists – sequences of numbers, character strings, or other lists. It provides them with operations for splitting lists apart and for making new lists by joining old ones. LISP programs also have the ability to modify themselves. However, the size and multiple nesting of function definitions in LISP are barriers to readability.

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 basis in this logic provides its distinctive flavour. PROLOG program is a series of declarative statements in logic which can be understood quite separately from considerations of how it will be executed. These declarative statements formulate PROLOG programs in smaller units and improve the readability. In addition, the built-in pattern-matching capability in PROLOG is an extremely useful device. PROLOG has the additional advantage of having a very powerful inference engine. PROLOG does, however, have certain deficiencies. For example, the use of built-in input/output predicates creates symbols that have no meaning in 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. A shell can be extremely useful in developing expert systems for a specific application, provided it is

well chosen. Shells do have limitations and disadvantages. Shells are inflexible and it may be difficult to fit them to non-standard problems and tasks [8]. As a result, a builder may use several shells and other tools, even in one application. Such proliferation may cause problems in training and maintenance. Shells also add an interface layer that requires its own resident interpreter. ES shells can be considered as limited programming environments. For examples, they may use only rule representation and only backward chaining.

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. Toolkits are based on two basic tools: Smalltalk and OPS. Working with toolkits makes ES more economically justifiable, especially when they are being developed on personal computers [8].

The metrics like applicability, abstraction, facilities and costs of hardware, software and training may be considered useful in the comparison process [1] 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.

With due consideration of the observations, we summarize that 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 fulfil all the requirements of the problem domain lays on an expert system. The programming facility, if any, of a toolkit is expected to fulfil 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 is 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.

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.

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

A decade back literature [9] would be adequate to explain the above head line. Bundy [9] provides a catalog of over 250 software products and AI techniques. Hopefully, this number should be over 500 at the end of 2002. 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.

5.2.9. 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.

5.2.10. 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 judgement 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 tasks. It is, rather little bit easier to suggest a general framework for the problem. Rothenberg [2] 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.

For example, 'contexts' dimension might demand five different tools suitable for five different contexts such as conceptualization, prototyping, development, fielding and operation / maintenance; leading to an unmanageable situation. But, however, pruning and prioritizing should make the situation manageable which again may depend on individual's experience and judgement. This might suggest another more simple framework. However, the matter is yet to be settled. But, it may be of general acceptance that such an evaluation and selection should primarily depend on the requirements the domain lays on an expert system and on the tool capabilities rather than tool features. This demands a thorough analysis of the problem domain, the

problem itself and the anticipated project including, even, the potential users of the proposed system.

5.3. ES-building tools' capabilities

It might be more important to focus on the capabilities of a tool, 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 a functionality. Users are generally interested in different capabilities (and it is also more convenient for less experienced end users) without knowing the technical features supporting those capabilities. Some representative potential capabilities and the corresponding supporting features are identified here as presented in table 5.1 which includes the suggestions of Rothenberg [2].

Table 5.1. Capabilities of tools with supporting features.

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

Let us now consider a large-hybrid-object-oriented toolkit - Level5 Object, as a potential tool at our disposal. Obviously, Level5 Object has certain good features and capabilities to be selected as an implementation toolkit for present problem domain. Moreover, an authorised version of Level5 Object is in use during last few years in the Expert System Laboratory of North Bengal University where this present system has been developed. Level5 Object is basically an object-oriented environment, so the object-oriented knowledge representation format chosen in the present work is strongly supported by it.

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.

5.4. Level5 Object

5.4.1. 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 [10].

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.

5.4.2. 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

These applications solve difficult scheduling problems that exceed the capabilities of conventional off-the-shelf solutions. Level5 Object's event-driven architecture allows one to create scheduling applications that adjust "on-the-fly" to the changing events of our day-to-day business activities.

Resource and constraint management

These applications juggle resources and resource constraints to find a best-fit scenario that can mean the business efficiency difference between one and one's competition. Modern business is moving to "just-in-time" operations in order to operate with the least cost and the highest productivity. Innovative organizations use Level5 Object as the critical decision-making link in these resource management applications.

Regulation compliance and conformance

Worldwide, government organizations are using Level5 Object to create "smart" forms systems that enforce consistent compliance with government regulations and ensure higher quality decision making by the work force. By codifying regulations and guidelines into business rules and embedding them into automated business process applications, many organizations have improved the quality and fairness of the services they provide.

Diagnostics

Level5 Object has the unique ability to resolve complex diagnostic problems across a broad range of industries. Level5 Object has proven itself as the quickest and easiest path to a solution, whether one is developing fault detection systems, real-time process monitoring and control applications, or business evaluation and decision support applications.

Client / Server

Level5 Object has built-in interfaces to all the common and important local and remote database servers on mainframes and mid-tier systems. Its robust and flexible support of the user interface environment on the client side, as well as its ease of use and development features, make Level5 Object an excellent tool to field client / server applications and cooperative processing.

5.4.3. Capabilities of Level5 Object

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

Arithmetic processing

Level5 Object provides arithmetic processing to the attributes of numeric type of different objects. The types of operators such as assignment, numeric, relational along with numeric functions and numeric value editor etc. are used for the purpose.

Built-in-functions

It provides on-system functions that operate on numeric, string, time, interval attributes. The typical functions are: Mathematical, Reference, Statistical, String, Time / Interval, Trigonometric, Value type conversion.

Certainty handling

Level5 Object provides a way to build reasoning with partial or uncertain information into a knowledge base. When confidence prompting is 'on', the user will assign a confidence value. In response to customer requests, confidence in Level5 Object can be expressed as a numeric value (-2...100). -2 indicates 'unknown' and -1 indicates 'undetermined'. Confidence values can also be assigned from within the rules of a knowledge base, which allows developers to quantify the degree to which they are confident in the accuracy of a conclusion or the degree of accuracy required to make a conclusion. This version supports product space confidence. With this technique, confidence is calculated by multiplying the confidence value of the antecedent and the value of the conclusion. Confidence management strategies other than product space can be developed with the help of confidence function.

Consistency checking

Level5 Object closely monitors all respects of knowledge base, it maintains complete referential integrity across rule, file and object management. Level5 goes beyond monitoring to prevent any attempt by a developer to change or accidentally delete classes, data, attributes, rules, instances or demons referenced elsewhere in a knowledge base.

Data type handling

It describes the type of information represented by an attribute of a class using attribute type. The attribute types are: colour, compound, instance reference, interval, multi-compound, numeric, picture, rectangle, simple, string and time. In addition, the attribute types may be of single value or an array of different sizes.

Documenting development

In Level5 Object, the process of documentation can be achieved using the display editor. Named rules and objects facilitate the documentation process in the system.

Explanation

The Level5 Object report system is a comprehensive explanation facility that provides complete access to the inferential reasoning process while running a knowledge base. In the case of the human mind, how a decision is reached is usually as important as the determination itself. Similarly, Level5 Object maintains full audit trails of how it arrives at its conclusions. Analytical reasoning facilities, such as session monitor, historical traces, single-stepping and breakpoint, enables developers and when necessary, end users to view all the current states of the inference engine; examine and change the state of any fact in a knowledge base; review the answers provided to Level5 Object quarries; 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.

Inference and control

Level5 Object can process information in a variety of ways: back chaining, parallel inference processing, dynamic agendas, rules, blackboard techniques and object-oriented programming. This versatility gives developers ultimate flexibility over the way in which data is presented and processed. It uses rules, demons, methods of WHEN NEEDED and WHEN CHANGED styles. These methods may contain some repetitive structures like IF...THEN.....ELSE; WHILE; DO... UNTIL and FOR.... TO. In addition, LOOP-statement is also available but not applicable to two those methods of WHEN NEEDED and WHEN CHANGED styles.

The class property INHERITS allows one to transfer the structure and behavior of a parent class to a child class. Consequently, a child class receives all of the capabilities of its parent classes and uses not only the attributes of the parent class, but also its methods, rules and demons. Therefore, one can reuse an application's code down the class hierarchy, increasing productivity and accuracy.

In Level5 object, a class can inherit from more than one parent. This feature, called 'multiple inheritance' allows one to create hierarchical chains of inheritance.

Integration

Direct database access enables Level5 Object to read and write to file types directly from within the knowledge base. Level5 Object supports dBASE III and III PLUS, Lotus 1-2-3 WKS and WK1, SQL and ASCII file formats. Using object-oriented database management techniques, developers build and maintain powerful

applications that are able to access large, heterogeneous data structures. By committing much of the underlying complexity of data access to system classes, developers can limit end-user access to only pertinent data and make the process of searching attribute values from databases virtually transparent. In addition, its DDE system class provides direct program-to-program communication capabilities. In this release Level5 object functions only as a client.

Internal access

Level5 Object provides different commands and facets for different parameter values at the beginning of a consultant session or when running an application e.g. RESIZE, INIT, REINIT etc. It also provides the source code of a knowledge base in a language what is known as Production Rule Language (PRL).

Knowledge acquisition

Level5 Object does not provide any facility for automatic knowledge acquisition for a system development.

Knowledge base editing

Level5 Object is equipped with five editors such as Objects editor, Database editor, Display editor, Methods / Rules / Demons editor and Windows editor. With these editors one can edit different parts of the total knowledge base.

Life cycle

It is in the nature of systems that share a common life cycle pattern. After a system has been in operation for a number of years, it gradually decays and becomes less and less affective because of the changing environment to which it has to adapt. For the time being it may be possible to overcome problems by amendments and minor modifications to the system but eventually it will be necessary to acknowledge the need for fundamental changes which demands a new system.

In Level5 Object, the required minor modifications in knowledge base in terms of objects, attributes, rules, methods, demons etc. are possible. It is also possible to delete any object or add new objects as when required. The system automatically takes care of the validity of any deletion or modification in the knowledge base. For a new system development the created objects can be used. This expedites the new system development.

Menus

The menu-driven facilities of Level5 Object makes it handy to the system developer as well as to the end users.

Meta-knowledge

Different facets of Level5 Object tell the inference engine how to process an attribute. Facets like BREAKPOINT, EXHAUSTIVE, REINIT etc. as well as Rules / Methods / Demons are used to control inference in an application.

Optimization

Separate compilation of source knowledge base is not required in Level5 Object. The object code is automatically created after addition of source text. It supports compiled execution for efficient application speed and size. One can expedite processing using the SMARTDrive concept. SMARTDrive is a disk-caching program provided with Windows. In addition, writing the software in C increases performance speed.

Presentation (I / O)

Considering I/O presentation, Level5 Object supports text, graphics, windows, forms, mouse and keyboard. Where mouse is not available, it runs fully using a keyboard.

Representation

Level5 Object is a fully object-oriented system. The objects are managed using objects editor. It also supports rules (single or group), methods (WHEN NEEDED and WHEN CHANGED) and demons.

5.5. 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.

5.6. Conclusions and discussion

It is important to select a proper tool for the development of an expert system. 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).

References

1. J. L. Alty. Expert system building tools. In Topics in expert system design. G. Guida and C. Tasso. (ed.). Elsevier Science Publishers B. V.; (North Holland), 1989. pp. 181-204.

2. J. Rothenberg. Expert system tool evaluation. In Topics in expert system design. G. Guida and C. Tasso. (ed.). Elsevier Science Publishers B. V.; (North-Holland), 1989. pp. 205-229.

3. P. Harmon (ed). The cost effectiveness of the major U.S. Expert Systems Building Tools. Expert Systems Strategies; vol.3, no.11, 1987. pp. 14-16.

4. I. Bratko. Fast prototyping of expert systems using Prolog. In Topics in expert System Design. G. Guide and C. Tasso. (ed.). Elsevier Science Publishers B. V.; (North Holland), 1989. pp. 69-86.

5. D. Prerau. Selection of an appropriate domain for an Expert System. The AI Magazine; Summer 1985. pp. 26-30.

6. A. B. Baskin and R. S. Michalski. An integrated approach to the construction of knowledge-based systems : Experience with advise and related programs. In Topics in expert system design. G. Guida and C. Tasso. (ed.). Elsevier Science Publishers B. V.; (North-Holland), 1989. pp. 111-143.

7. W. Mettrey. A comparative evaluation of expert system tools. IEEE Computer; February, 1991. pp. 19-31.

8. E. Turban. Expert Systems and Applied Artificial Intelligence. Mcmillan Publishing Co. New York. 1992. pp. 453-470.

9. A. Bundy (ed). Catalogue of artificial intelligence tools. Second Revised ed., Springer-Verlag; N. Y.,1986.

10. LEVEL5 OBJECT for Microsoft Windows Getting Started Guide (Release 3.0). Information Builders, Inc.; USA, 1993.