User Interfaces for Knowledge Based System Tools

John Kingston

AIAI-TR-101

March 1992

This paper was published in the proceedings of the British Computer Society workshop on "User Interfaces for Expert Systems", which was held on 11-12 March 1992 at the RHS Conference Centre, Vincent Square, London.

Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

Abstract

The development of user interfaces for the developers of knowledge based systems (KBS) tools has mirrored the development of KBS themselves. In the early 1980s, the state of the art software for KBS was Lisp and its debuggers. As rapid prototyping came into vogue in the mid 1980s, so the interfaces of KBS supported rapid learning, development and testing of knowledge bases. With the growth of the KBS market, the shortage of trained knowledge engineers has led to a desire for tools which are simpler to use; user interfaces have played an important part in the resulting move towards higher-level tools. There has also been a general move towards standardisation of interfaces, which the user interfaces of KBS tools have followed; this has opened up the field for KBS to communicate with other user interface packages.

Possible future directions, including computer-aided knowledge engineering tools and tools which automate the knowledge acquisition process, are also discussed.

1 Introduction

Knowledge based systems (KBS) first emerged as a viable commercial technology in the early 1980s with systems such as MYCIN, DENDRAL and PROSPECTOR. In those early days, the user interfaces for developing KBS were those provided by Lisp; the state of the art software for interactive KBS programming was Lisp and its associated debuggers. As time has gone on, the process of developing KBS has changed considerably, and the user interfaces for developers have changed accordingly. This paper traces those changes, from rapid prototyping through to higher level tools and standardisation, and looks at the user interface facilities provided for each approach. It also looks at possible future directions, and at the user interfaces of the tools of the future.

This paper is based on experience in developing KBS and evaluating KBS tools gained at the Artificial Intelligence Applications Institute of the University of Edinburgh (AIAI). AIAI has designed and implemented KBS for industrial and commercial clients since it was founded in 1984. During this time it has evaluated both commercial and experimental KBS development tools.

2 Rapid Prototyping

One characteristic of KBS which makes them different from conventional computer systems is that the knowledge they encode is often acquired and structured during a KBS development project, rather than beforehand. The effects of this one difference have been far-reaching. The need to start to implement a KBS while its knowledge was still being acquired led to an emphasis on tools which allowed the knowledge base to be updated and altered easily. When it was realised that this initial prototype KBS could be a useful tool in knowledge acquisition, the dea of "rapid prototyping" as a KBS development method arose. Rapid prototyping was *the* way to develop KBS in the mid-1980s.

Support for rapid prototyping became a key concern in the development of many KBS tools. This had, and has, considerable implications for the user interfaces of KBS tools. Interfaces were designed

- to minimise the learning time for the facilities of KBS tools
- to make it possible to create rules and/or objects quickly
- to make the knowledge base readily accessible
- to make the knowledge base easy to debug

The degree of success achieved varied between different tools. The rest of this section illustrates some of the techniques applied to each of the four areas noted above.

2.1 Minimising learning time for KBS tools

The ease of learning a KBS tool depends primarily on the amount of functionality it offers. Many people find Crystal very easy to use, for example, because it only supports one type of knowledge representation (backward chaining rules), and it provides an interface to make it easy to write such rules. However, when comparing tools of similar functionality, there is little argument that the easiest KBS tools to learn are the tools whose basis is in object-oriented programming as opposed to rule-based programming. KEE and Goldworks were the earliest tools in this category to achieve widespread commercial use; more recent examples include KAPPA-PC and ProKAPPA.

The key to the ease of learning appears to be the consistency of the interfaces offered by these tools. In Goldworks, for example, almost everything the system does is channelled through object-oriented programming. As a result, it is possible to offer the same object-like interface for almost every function of the tool. To take an example, communication with an external spreadsheet file is carried out by creating an object to represent that spreadsheet. This object has four slots – START-ROW, START-COLUMN, END-ROW and END-COLUMN – which are filled in to indicate which portion of the spreadsheet is to be read. The object-level interface even extends to initiating spreadsheet access; there is a slot in the object named GO, and when the user enters :YES in that slot, the spreadsheet file is accessed. The value(s) read and any error messages are stored by Goldworks in further slots of the object.

While this style of interface can be quirky at times, it makes it fairly simple for a KBS developer who is familiar with one part of the tool to use another part of the tool. Contrast this with Knowledge Craft, which combines versions of OPS5, Prolog and an object-based language in one system. This requires a KBS developer to learn three different syntaxes in order to use all its features.

Ease of learning also depends on how 'natural' the interfaces are; that is, whether the user interface matches the user's "natural idiom". The "natural idiom" is the way a user thinks about the problem (cf. Stelzner and Williams [10]). Stelzner and Williams argue that people's "natural idiom" is frequently graphic, and they therefore argue for graphical specification of a knowledge base. The tool which makes the most effort to provide graphical specification for knowledge bases is KEE (see section 3.1 for more details). This combination of consistency and naturalness means that the time required for developers to become familiar with all the facilities of KEE is roughly half the time required to reach a similar stage in Knowledge Craft [1].

2.2 Making it possible to develop rules and objects quickly

Most shells and toolkits offer facilities to smooth the creation of rules and objects. Crystal, for example, offers a rule editor with certain macro functions, and a dictionary of variables, available by pressing particular function keys . Most of the toolkits, such as ADS, Nexpert Object, KEE and KAPPA-PC provide similar editors, or templates, for the creation of rules and objects. However, once a KBS developer is reasonably familiar with a tool, it is often quicker to write rules and objects using a text editor. When this happens, the toolkit needs to be able to recompile individual rules and functions incrementally. The "big three" toolkits (Inference ART, KEE and Knowledge Craft) are all able to perform incremental compilation from a standard text editor (such as EMACS or VI), but few other tools can do so. Indeed, some tools encourage developers to save knowledge bases in binary format, which requires developers to use the built-in editors; this can be irritating if a KBS developer is very familiar with a certain text editor.

2.3 Making the knowledge base readily accessible

Most KBS tools provide an interface for browsing the contents of the knowledge base, usually based either on pull-down menus or function keys. It is normal to be able to view the rules, facts or objects in the knowledge base either individually or side by side. Most tools which allow the development of hierarchies of objects also provide a window for examining object hierarchies. This is often a useful tool, particularly if can be used interactively to access further information about objects.

[1] This estimate is extrapolated from experience with a number of visitor projects at AIAI. It should be noted that few of these visitors had attended the tool vendors' training courses.

The tools which really score well on ease of access to the knowledge base are the (few) tools with good cross referencing of knowledge; for example, from browsing an object, a developer can browse a rule whose conclusion affects that object, and can then move to another object which is matched by the conditions of the rule. Contrast this with the situation where a developer browses a rule, finds which object matches its conditions, and must then return to the root menu and work through several other menus in order to view that object. ART probably offers the best knowledge base cross referencing of any tool; almost all its menus and debugging interfaces can be used to initiate further traversal of the knowledge base, instead of having to return to a root menu.

2.4 Making the knowledge base easy to debug

The functionality available for debugging a knowledge base is an important feature of any KBS tool, and user interfaces have been a key feature in this process. A number of toolkits, such as KEE, Knowledge Craft, and KAPPA-PC support graphical traces of rule firing; many toolkits support mouse-based access to the partial matches of a rule; and many shells and toolkits support multiple windows in which textual information about the knowledge base can be accessed and then compared with other parts of the knowledge base.

The key factor which makes user interfaces a valuable debugging tool is providing rapid access to information, whether it be information about the knowledge base, as described in the previous section, or information about rule firings or partial matches. As an illustration, a project which was carried out at AIAI using ART came across a bug in ART's cross-referenced menu system [2]. This bug forced us to avoid using the menu system. The result was that errors in the knowledge base which previously took 5 minutes to track down and fix were taking half an hour instead.

3 Higher level tools

So far, this paper has concentrated on toolkits for building KBS. The user interface world also has toolkits, for the construction of user interfaces; Suntools and Xtk are examples. However, it has been found that these toolkits are too low-level for many, and so a number of different attempts have been made to produce higher level tools such as declarative languages, tools for specifying a user interface graphically, and application frameworks such as MacApp [3]. The parallel with KBS is obvious, and the KBS world has responded to the demand for tools which are easier to use. This section describe two different responses.

[2] The bug, which was in ART 3.0, has been fixed in later versions of ART.

3.1 Graphical specification of a knowledge base

As stated previously, KEE provides good facilities for graphical specification of knowledge bases. The facilities provided in KEE are as follows:

- KEEPictures, which are easy to define graphics, represented internally by an object which is created automatically
- ActiveImages, which are interactive KEEPictures; when a setting is changed on a graphic using the mouse, the underlying object is also updated, and vice versa.

In certain circumstances, graphical specification of a knowledge base can be very useful. Stelzner and Williams [10] give examples of using both KEEPictures and ActiveImages to build interfaces for KBS which model a life support system; analyse strains of genes; model a feedwater system for a nuclear power plant; simulate a factory floor; and perform project management. It is noteworthy, however, that all of these examples (with the possible exception of project management) require modelling of complex real-world entities. It is likely that graphical specification of a knowledge base is effective for tasks of this sort, but less effective for other tasks.

3.2 Application-specific tools

Application-specific shells tend to be built on top of existing shells or toolkits, but provide a number of facilities specifically designed to support the construction of KBS in a particular application sector. Some early attempts to add financial functions to certain shells, (such as Crystal City, an extended version of Crystal) were not very successful; the application-specific tools did not really come into their own until their user interfaces were also customised to particular applications.

One of the most popular applications for these tools has been the support of help desk operators. This is essentially a diagnostic task; the purpose of the KBS is to diagnose the cause of faults to an extent where simple faults can be solved by the help desk operator (while using the KBS), and more complex faults can be directed to the correct expert. There are several help desk-specific KBS tools on the market, such as Path Builder, Mahogany Help Desk, The Help Desk Assistant, and others. Many of these tools provide a *decision tree* interface for the development of the knowledge base. The KBS developer draws out a decision tree of faults on the screen, using a mouse and menu interface, and this tree is automatically translated into an underlying formalism (which is usually backward chaining rules). An example of such an interface is given in Figure 2.

Figure 1: Examples of KEE's ActiveImages

One major advantage of decision tree interfaces, and other high-level interfaces for application-specific tools, is that the KBS developer could be an expert in diagnosis, rather than an expert in KBS. This is possible because many experts can understand a decision tree: it appears to be their "natural idiom" . However, decision tree interfaces share some of the disadvantages which are associated with high-level user interface toolkits [8]: they limit the range of the knowledge bases which can be created, and they cannot support certain knowledge base functionality.

Other application-specific shells which are on the market include a number of shells for general diagnosis such as Testbench, shells for real-time network monitoring such as DANTES [14], and case-based reasoning shells such as CBR Express and IDOTS (the Intelligent System for Design of Communication Systems) [4]. The developer's interface of CBR Express is interesting, because it consists solely of forms asking for English language descriptions of previous problem cases, their symptoms and their solutions. CBR Express works by pattern matching descriptions of cases against English language descriptions of faults; the results of a pattern match are used to determine symptoms which would be worth investigating, and the likelihood of particular faults. Other application-specific shells which are on the market include a number of shells for general diagnosis such as Testbench, shells for real-time network monitoring such as DANTES [14], and case-based reasoning shells such as CBR Express and IDOTS (the Intelligent System for Design of Communication Systems) [4]. The developer's interface of CBR Express is interesting, because it consists solely of forms asking for English language descriptions of previous problem cases, their symptoms and their solutions. CBR Express works by pattern matching descriptions of cases against English language descriptions of faults; the results of a pattern match are used to determine symptoms which would be worth investigating, and the likelihood of particular faults.

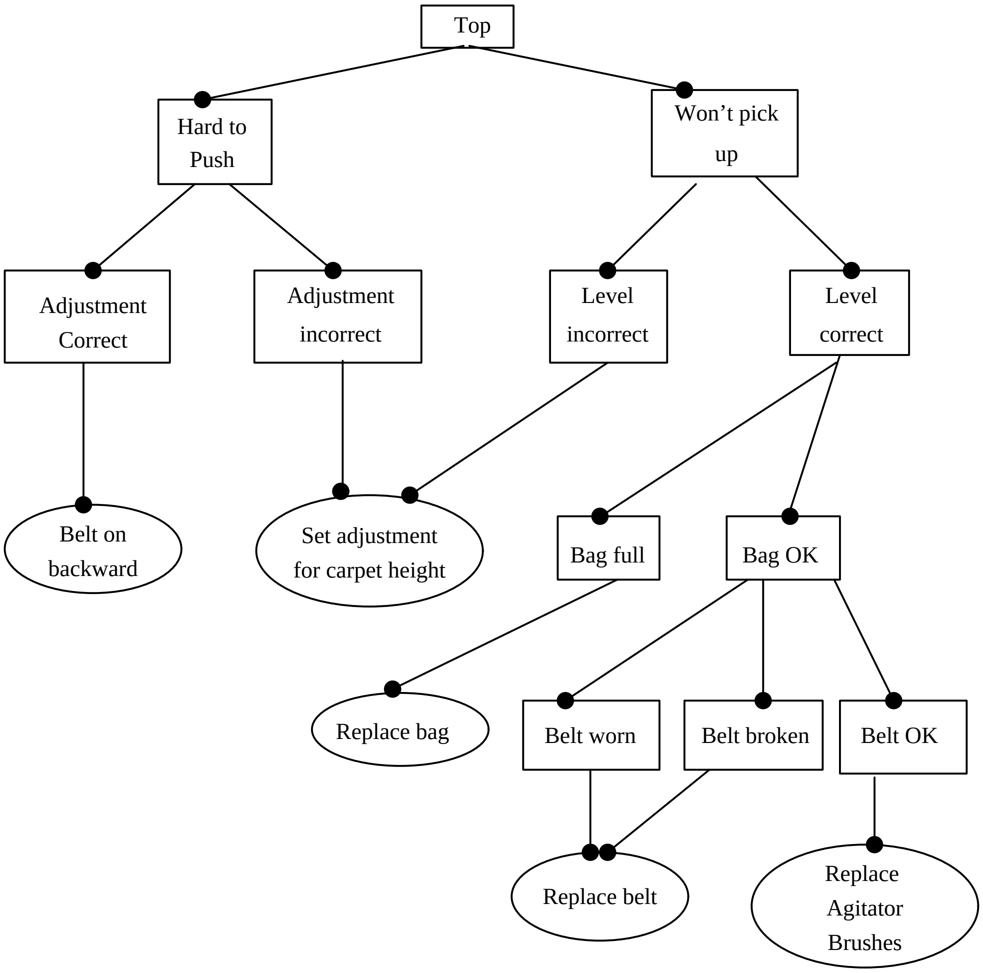

Figure 2: A decision tree for the Mahogany Help Desk, showing how to diagnose and fix an ineffective vacuum cleaner (from [5]).

4 Standardisation

In the last few years, there has been a move towards standards for user interfaces by software and hardware manufacturers. Current "standards" include Microsoft Windows 3 for applications running on IBM-compatible PCs and X-Windows for workstation users. While these standards are *de facto* rather than *de jure*, many KBS tools are being ported to the "standard" environments; many PC-based tools now run under Windows 3, while on workstations most major toolkits have been developed or ported to run under X-Windows.

The greatest benefit of this move has been easier integration between KBS tools and other user interface packages. For example, the X-MATE system [6], which was implemented in KAPPA-PC on a PC running Windows 3, originally used Asymetrix' Toolbook for its dynamic form-based user interface. This was accomplished using the client-server architecture provided by the Dynamic Data Exchange facility, which is available as part of Windows 3. A more recent innovation is the delivery of PC-based KBS tools as a collection of dynamic link libraries (DLLs), which allow all or part of a KBS tool to be compiled into the same executable file as another package. Examples of KBS tools which are provided as DLLs are ART-IM (Windows version), ECLIPSE, and KAPPA-PC (version 2.0). The availability of Open Systems, a client-server architecture for workstations which is fast becoming a standard (cf. [1]), should see similar possibilities for integration in the workstation world.

5 A Future Look

It seems reasonable to assume that, just as user interfaces have followed and supported the development in the KBS field to date, they will continue to do so in future years. Some future directions in knowledge engineering, and associated tools, are discussed below.

5.1 Automated knowledge elicitation tools

Automated knowledge acquisition elicitation tools are designed to eliminate the knowledge engineer from the process of KBS development, or at least to reduce the role he plays. The idea is that by eliciting knowledge directly from a domain expert and structuring the resulting knowledge automatically, there is much less need for a knowledge engineer; instead, the expert becomes the primary KBS developer. Such tools could be classified with the high-level tools described in section 3. Automated knowledge *elicitation* tools are those tools which extract knowledge from a human expert; tools which extract knowledge from a database of case histories have little need for user interfaces, and so are not discussed here.

The user interfaces of automated knowledge elicitation tools ask the expert a series of questions, and then structure the resulting information into a classification hierarchy, a decision tree, or statistically-derived clusters, depending on the tool being used. The resulting structured information may be shown on the screen, as in ProtoKEW [12], which offers a range of automated knowledge elicitation tools. Other tools, such as Nextra (an automated repertory grid tool [11]) and the KnAcq (which encodes a novel approach to knowledge elicitation, involving asking for exceptions to some general rules), do not show the resulting structure via the user interface; instead, it is converted into rules in the syntax of a particular tool. Nextra operates as a front end to Nexpert Object, while the KnAcq can produce rules in the syntax of a number of different tools.

There are two disadvantages with automated knowledge acquisition tools. The first is that they only work really well if the problem is concerned with classification or diagnosis. Secondly, the rules produced may be verbose, or hard to understand, which is largely due to the automated knowledge acquisition methods being primarily aimed at eliciting categories rather than rules. A creditable attempt to overcome this problem has been Mark Musen's approach to *model-based elicitation* [9]. Musen's work now spans three types of tool: knowledge based systems, such as ONCOCIN, which is a KBS for advising on treatment of cancer patients; automated knowledge elicitation tools, such as OPAL, which is designed for creating ONCOCIN-like systems; and PROTEGE, which is tool for creating OPAL-like systems.

The key difference between OPAL and tools such as Nextra is that OPAL is application-specific; it knows that its task is to elicit correct drug regimes, to ask for rules concerning medical diagnosis, and so on. OPAL contains some medical information, such as different types of drugs. Musen has paid a great deal of attention to the user interface, and the "natural idiom", in all his tools; the result has been an interface based on menus and forms in PROTEGE and OPAL, while ONCOCIN's primary interface resembles the traditional flowsheet which doctors previously used for data collection. The key difference between OPAL and tools such as Nextra is that OPAL is application-specific; it knows that its task is to elicit correct drug regimes, to ask for rules concerning medical diagnosis, and so on. OPAL contains some medical information, such as different types of drugs. Musen has paid a great deal of attention to the user interface, and the "natural idiom", in all his tools; the result has been an interface based on menus and forms in PROTEGE and OPAL, while ONCOCIN's primary interface resembles the traditional flowsheet which doctors previously used for data collection.

Figure 3: An example of the user interface of the OPAL system (from [9])

5.2 Computer-aided knowledge engineering tools

There is a strong emphasis on methodology in the KBS field at the moment. Customers want KBS to be reliable, verifiable, and maintainable; it is hoped that the use of methodology will provide all these benefits, as well as guiding the knowledge engineer in the acquistion and structuring of knowledge.

In the software engineering domain, the introduction of methodology incited the development of a number of tools, which were intended to support the development of software using certain methods. These became known as computer-aided software engineering (CASE) tools. By analogy, it is likely that the KBS software suppliers will soon be offering a number of computer-aided knowledge engineering (CAKE) tools. A few such tools have already been developed, such as Shelley [2], which is a knowledge acquisition and structuring tool based on the KADS methodology for building KBS, and Acquist, which is part of KEATS [7]. A fuller list of tools is available in [13]. Shelley, which was originally developed by the Department of Social Science Informatics at the University of Amsterdam, offers a number of facilities to support the use of the KADS methodology, including

- An interactive editor for structuring fragments of interview transcripts into a concept network
- "Data-flow" diagrams to represent the models used by the KADS methodology
- A library of task-specific "interpretation models", which are a key feature of the KADS methodology

Figure 4: An example of the user interface of Shelley (from [2])

6 Conclusion

The development of user interfaces for KBS tools has followed the development of KBS tools themselves. In the mid-1980s, the demands of rapid prototyping produced user interfaces for KBS tools which made the knowledge base easy to develop, easy to access, and easy to debug. The 1990s saw a move towards high-level KBS development tools, whose success was crucially dependent on their user interfaces. Another move has been towards standardisation, with many KBS tools now running uder X-Windows on workstations or Windows 3 on IBM-compatible PCs; the benefits for integration with other user interface packages have been noticeable. The future holds a number of possibilities; KBS may be developed using automatic knowledge elicitation tools or computer-aided knowledge engineering tools. The user interfaces of these tools will depend heavily on the underlying methods.

References

[1] . USL seeks end to Unix incompatibility. *Computing*, page 7, 13 February 1989.

[2] A. Anjewierden and J. Wielemaker. Shelley – Computer Aided Knowledge Engineering. In *Current Trends in Knowledge Acquisition*, pages 41–59. IOS Press, 1990.

[3] B.A. Myers. State of the Art in User Interface Software Tools. In H.R. Hartson and D. Hix, editor, *Advances in Human-Computer Interaction*, volume 4. Ablex Publishing, 1991.

[4] C. Tsatsoulis. Case-based design and learning in telecommunications. In *2nd International Conference on Industrial & Engineering Applications of AI and Expert Systems*, volume 2, University of Kansas, 1989.

[5] Emerald Intelligence. Mahogany Help Desk demonstration disk. Emerald Intelligence can be contacted at 3915-A1 Research Park Drive, Ann Arbor, MI 48103, USA.

[6] J. Kingston. Knowledge Based Systems in the UK Financial Sector. In *Proceedings of "An Analysis of Expert Systems in the European Banking Industry"*, Milan, 7-8 March 1991. This paper describes the X-MATE development project and knowledge base architecture. It is also available as AIAI-TR-87 from AIAI, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN.

[7] M. Eisenstadt, J. Domingue, T. Rajan and E. Motta. Visual Knowledge Engineering. *IEEE Transactions on Software Engineering*, 16(10), October 1990.

[8] M. Linton, J. Vlissides and P. Calder. Composing User Interfaces with InterViews. *Computer*, 22(2):20, February 1989.

[9] M. Musen. *Automated Generation of Model-Based Knowledge Acquisition Tools*. Morgan Kaufmann, 1989.

[10] M. Stelzner and M. Williams. The Evolution of Interface Requirements for Expert Systems. In J.A. Hendler, editor, *Expert Systems: The User Interface*, pages 285–306. Ablex Publishing, Norwood, New Jersey, 1988.

[11] Neuron Data. Nextra Overview. Neuron Data Inc., 34 S. Molton Street, London W1Y 2BP. 071 408 2333.

[12] Han Reichgelt and Nigel Shadbolt. ProtoKEW: A knowledge based system for knowledge acquisition. In D. Sleeman and O. Bernsen, editors, *Recent Advances in Cognitive Science:*. Lawrence Earlbaum Associates.

[13] Robert Inder and Ian Filby. Survey of Methodologies and Supporting Tools. Technical Report AIAI-TR-99, Artificial Intelligence Applications Institute, University of Edinburgh, 1991. This paper was presented at the BCS SGES

workshop on KBS Methodologies, held at the RHS Conference Centre in London on 3 and 4 December 1991.

[14] Van Cotthem H., Mathonet R., & Vanryckeghem L. DANTES: An expert system shell dedicated to real-time troubleshooting. In D.J. Sassa, editor, *Proceedings of the International Communications Conference 1987*. IEEE, June 1987.