Expert Agents in Knowledge Based Systems

MAURICE D. MULVENNA, JOHN G. HUGHES

Department of Information Systems, University of Ulster at Jordanstown,
Shore Road, Newtownabbey, Co. Antrim BT37 0QB
email: {cbma23,jgh} @uk.ac.ulster.ujvax

ABSTRACT
The concept of *expert as agent* is introduced in this paper. The expert resides in an object-oriented environment that models a problem domain, and has access to particular rules sets, which can infer expert level knowledge about the domain. The purpose of the expert as agent is to enhance the **usability** of resulting *knowledge systems*, and to promote *technology transfer*. A prototype developed in the domain of aeronautical engineering is used to present the ideas outlined in the paper.

1 INTRODUCTION
The scope of software tools and programming paradigms currently available are a great advancement on those available when expert systems technology first became accessible to many with the coincidental arrival of rule-based systems and low-cost personal computing in the early 1980's.

Knowledge-Based System (KBS) development environments now in use have become much more flexible than their 'shell' predecessors. Currently such systems are capable of addressing large problem areas, and provide a comfortable mixed environment, combining user interface design, KBS techniques, and proven coupling with external software, especially 'efficient' languages such as C / C++, and databases.

Object-Oriented Programming (OOP), with its roots in simulation, has been accepted as a powerful aid to information systems development. This is primarily because the software developers recognise the ability of OOP to model entities and events naturally. This benefit is particularly appreciated by developers who use object-oriented KBS, because of the resultant increase in the speed of prototyping applications. Indeed, it is primarily for this reason that object-oriented KBS have been recommended as the most suitable tool in which to prototype *any* application [Graham91].

A logical extension to the use of OOP is to make objects persistent. In effect, to create object bases. The term object-oriented databases is not truly applicable as object bases may contain complex inter-related and composite objects as opposed to simple data. The ability to store and manipulate objects in multi-user environments provides greater scope to developers using OO software. Contemporary OO KBS environments such as ProKappa do not provide the facilities needed for object management. Object-orientation is pervading most areas of computing today. It is common in:

- Languages - C++ / Smalltalk V
- KBS tools - ProKappa / NExpert Object
- User Interface Development - ObjectVision / Visual Basic
- Databases - ONTOS / ObjectStore
- Multimedia & Computer-Based Training (CBT) - KnowledgeProWin

Object-oriented KBS and object bases give the KBS developer the tools to build large, complex knowledge bases containing rules (rules can be objects too!) and objects. However, some method of combining expertise (in the form of rules) to objects is needed, to bind the rule-based formalism to object-orientation in such a way that both can be used easily. A way to do this entails imbuing objects with knowledge.

2 MERGING OF KBS WITH OBJECT-ORIENTED SYSTEMS

The merging of KBS with OO systems has been driven by those in the KBS camp, and by those who perceive object-orientation as a 'good thing'. The two camps are not mutually exclusive. The merging of these technologies is best exemplified by looking at two particular areas; embedding of OO concepts into KBS development software, and the coupling of KBS to object bases.

2.1 Embedding OO in KBS

Despite many vendor's claims, the integration of OO into KBS has not been completed. Some systems, for example, ProKappa (from Intellicorp Inc.), provide the developer with a rich working environment that mixes object-orientation, production rules, and procedural programming. However, ProKappa does have the following problems:

- The rules violate the encapsulation tenet of object-orientation. That is, they can directly access the attributes of an object, thus bypassing the public methods of such objects. This can be programmatically overcome, by providing *accessor* methods for all attributes - at the expense of the resulting rule syntax.

- Rule are not objects. The rules exist as separate entities. Again, this can be worked around by providing rule group objects that have attributes with multiple cardinality. These attributes contain lists of relevant rules. Methods may be written for such objects, and in a resulting system, particular rule groups can infer knowledge as a result of messages being passed to them.

The above example is typical of such systems, where a particular aspect of object-orientation is missing. Such problems can often be overcome, usually with a loss of coherence to the development process.

2.2 Coupling of KBS to Object Bases

KBS have been linked to (relational) databases using some form of coupling mechanism [Torsun89], but both KBS and DBMS have evolved rapidly over the past decade. The frame-based systems [Minsky75] that AI researchers have used for many years now have their counterparts in the database world, in OO data models. Although production rules may be used to represent knowledge, they may now be encapsulated within objects, and activated upon by methods. Most of these systems are still in the research laboratories.

A system which doesn't really fit the OO tag, but is worth mentioning, is the MegaLog system (from ICL Ltd.). This provides tight coupling between a Prolog-like logic system, and persistent data storage, using BANG file structures to store the data.

3 THE AGENT-BASED PARADIGM

An extension of the object-oriented paradigm is Agent-Oriented Programming (AOP) [Shoham93]. Yoav Shoham summarises AOP as:

> The state of an agent consists of components such as beliefs, decisions, capabilities, and obligations; for this reason the state of an agent is called its *mental state*. The mental state of agents is described formally in an extension of standard epistemic logics: besides temporalizing the knowledge and belief operators, AOP introduces operators for obligation, decision, and capability. Agents are controlled by *agent programs*, which include primitives for communicating with other agents. In the spirit of *speech act theory*, each communication primitive is of a certain type: informing, requesting, offering, and so on.

This concept of personification of an aspect of an information system is not new. *Actors* have been used in AI before [Hewitt85], and it could be argued that blackboard systems [Nii86] fit in this category. Blackboard systems originated in AI, and in such systems, co-operating experts 'pass on' their inferred knowledge to a blackboard, a kind of bulletin board where new knowledge is available for use by all other experts.

But it is the coupling of the concept of personification with object-orientation that has exciting implications for KBS and user interface design. In this paper we introduce the concept of *expert as agent*.

3.1 Our Working Definition of Agent Based Systems

Currently, the ideal of AOP as defined by Shoham is set aside for our systems. It is expected that part of Shoham's work, and aspects of speech act theory will be incorporated into future systems. However, what is needed immediately is the means to relate the knowledge of a domain (in the form of production rules) to the more static knowledge (best stored as objects, ultimately as persistent objects in an object base) about that domain. We feel that the agent metaphor provides that link. The agent is given the title expert because they *are* the experts - they reason and infer things about complex knowledge.

In the prototype system that has been developed, only one expert is present. This related naturally to the problem domain of aeronautical engineering, where knowledge was elicited from only one expert. If employing more than one expert as agent in a system, there are two main research routes to follow, one complex, the other relatively simple to implement:

- *co-operating experts*. This is the complex path, where components of Shoham's work must be implemented. An alternative way to accommodate co-operating experts is to fall back on the idea of blackboard systems.

- *expert opinions*. This approach is analogous to database views, where the user is presented with different views of the data as a result of calling various sub-programs in a database system. In a KBS, expert opinions would not co-operate, and could infer different knowledge commensurate with their role. Although quite simple, this concept has great potential when applied to KBS. Different levels of expertise could be modelled as expert opinions. Systems that provide user modelling could decide which expert opinion most suited the user.

4 THE TUCANO DEMONSTRATOR

The Tucano System was developed as a demonstrator, to publicise KBS within Shorts Plc. It provides a trouble-shooting function by assisting the technicians in diagnosing faults and incorrect connections when assembling aircraft. This releases the engineers to concentrate on their main tasks.

4.1 Problem Outline

The Shorts Tucano aircraft is a single engined (turbo-prop) trainer aircraft, in use world-wide. When the aircraft is being assembled, and prior to flight tests, the engine is connected to various electronic and mechanical components, and readings are taken at various engine running settings. Over a period of time the total number of readings that need to be taken to give a good diagnostic capability has been narrowed down by the engineers working on the aircraft to eighty-one. These readings have been compiled onto a paper data sheet, which is used by the technicians for every aircraft tested. When building the aircraft, some problems occur more frequently than others, and the technicians usually identify such faults immediately. But rarer faults are more difficult for the technicians to trace, and frequently the engineers need to be consulted. The engineers draw on their more in-depth knowledge of the aircraft to solve the rare and difficult faults, but their availability is not always guaranteed.

The Tucano Engine Ground Running Data Sheet System is a diagnostic system which assists technicians by identifying faults when the engine is first fitted to the airframe. It is designed to replace the manual paper data sheet, and free the time that the aircraft engineers spend in identifying faults.

4.2 Design Goals

In designing the Tucano system, it was soon apparent that more than a diagnostic KBS was required. The technicians needed visual re-enforcement when it was necessary for a failed component to be changed, and various manuals had also to be available in the system for consultation. In addition, some faults could be common to particular batches of engines, so an additional facility was required to compare datasheets of engine performances. Consequently, the Tucano System was designed to operate in four modes. As a:

- *Computer-Based Training (CBT) System*. To be useful, the system should provide a learning environment where the technicians can increase their skills in identifying faults.

- *Validation tool*. All the data entered into the Tucano system needs to be validated.

- *KBS that models the engine*. These rules encapsulate the aircraft engineer's problem-solving expertise, and govern the diagnostic process.

- *Specific engine test manager*. The technican must be able to load previous test data pertaining to older engine trials, and to perform the modelling and diagnostic tests on this data. This can be viewed as an extension to the CBT mode, as it provides the technician with a library of information with which to compare and contrast results.

The four modes are designed to operate together, providing a failsafe environment where faults are identified, and the technician's level of understanding of the underlying processes is raised.

4.3 Methodology, Elicitation & Design

Early knowledge-based systems were built with *ad hoc* methodologies where a part of the system was prototyped to illustrate the intended functionality of the finished system. Contemporary KBS methodologies are more complex. The most popular is KADS [Wielinga92]. Unfortunately KADS is large and unwieldly, and unsuitable for use in small to medium size software developments. This problem has been partly addressed by pragmatic KADS [Kingston92], yet the core aspects of KADS remain complex. In addition KADS does not refer to the use of prototyping, or, in detail, to the implementation phase. Prototyping KBS systems in an OO environment such as ProKappa is very useful, both to the developers of the software, and to end users. It can also provide positive feedback to the backers of KBS projects.

So what methodology could be employed? The design methodology selected for the Tucano project was based on simple object modelling concepts. This object model base provided a foundation to represent the static components of the system. The expert as agent, as a special kind of object, governs and mediates the use of the expert knowledge, represented as sets of rules. Thus, an agent represented internally in the system acts as focus when combining knowledge as rules with data / information as objects.

Knowledge elicitation in the problem area was conducted primarily through 3 phases of interviews, and by reference to specialist manuals, authored by the Tucano engineer. The interviews were conducted over a period of six weeks (Table 1), and each lasted 2-3 hours. Usually four people were present; two knowledge engineers, one Tucano engineer, and an observer from Shorts Plc.

	Interviews	Type	Outcome
Phase I	3	Open	Problem understanding
Phase II	4	Structured	OO Prototype
Phase III	3	Structured	Rule base

Table 1. Knowledge elicitation through interviews.

Knowledge elicitation began with a series of three open interviews. The interviews were transcribed, edited, and used as the basis of discussion for the next series of four interviews, which were more structured, with the goals and objectives of each meeting agreed at the beginning of interviews. From this second stage, a prototype was produced, which employed a user interface not radically different from the paper data sheet familiar to the technicians. The third stage of three interviews focused on rule elicitation. This progressed through the use of example problems, and the introduction of *what if* scenarios.

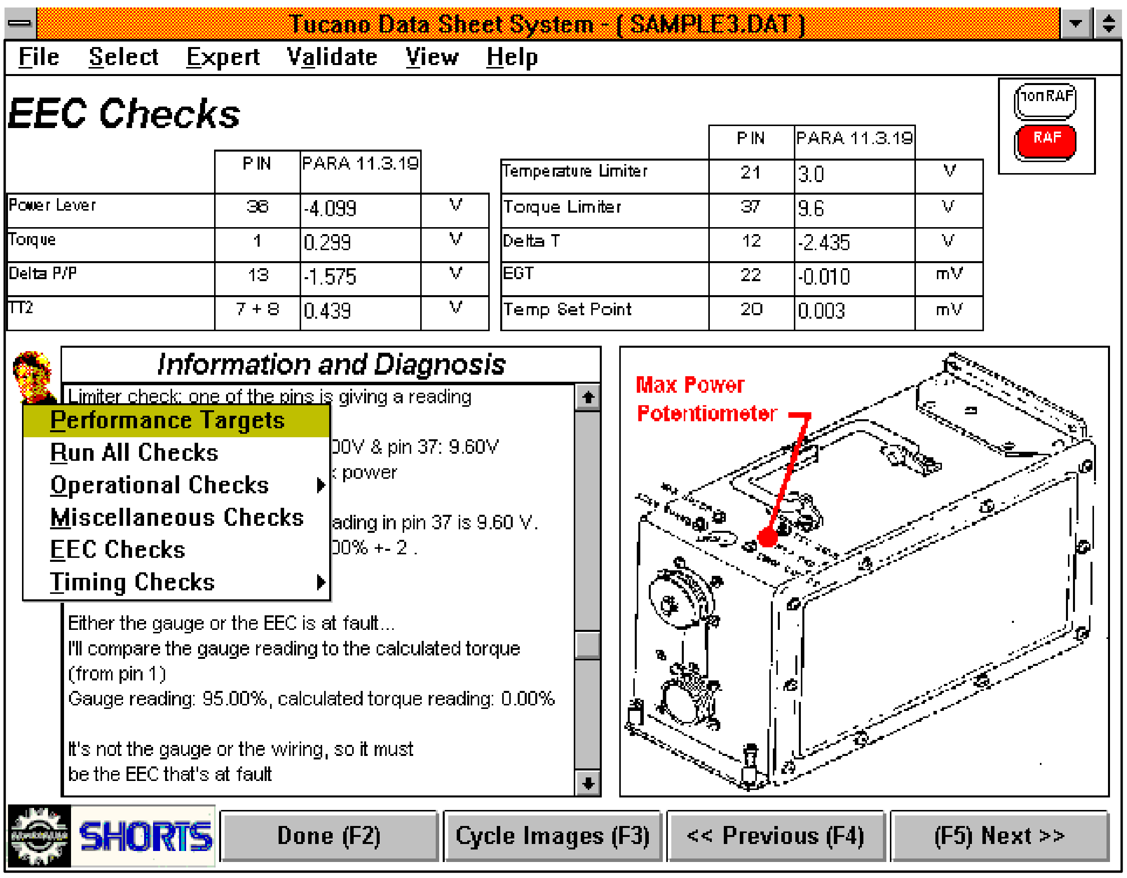
All of the relevant entry points on the data sheet were built into the system as objects. Each object could be referenced through a Unique IDentity (UID) label, and had a direct link to its User Interface (UI) component. The four operation modes were incorporated into the design of the system. The **CBT System,** providing various context-specific information, was made available to users by clicking with the right mouse button on any of the yellow-marked areas of the data sheet. The technician can access the relevant manuals, which depict information in both textual and graphical form. A standard help system provided search facilities. **Validation** may be carried out on some system parameters using the Validate menu, although the main workload of validation is carried out by constraint rules, initiated through an Expert menu. The validation rules are low-level rules which enforce constraints in the form of equations. For example, there is a simple relationship between the engine RPM expressed as a percentage, and RPM as a frequency. Some of the constraints have been derived from empirical tests, for example, the equation representing oil temperature versus oil pressure. The constraint-enforcing rules capture entries with incorrectly-placed decimal points, for example, and encourage technician's understanding of what the values actually represent. The **diagnostic KBS** component has been designed to use the expert as agent metaphor to relate rule sets to particular objects. The **engine test manager** enables the user to save the data entered, as objects, in binary file format, for later retrieval.

4.4 Functionality and the User Interface

The user interface (figure 5) of the system was carefully designed to facilitate the four design goals. In addition to the usual Graphical User Interface (GUI) components, there is a pictorial representation of the expert on-screen. The user can interact with this icon by either clicking on it, or with the user of a touch screen, by simply touching the expert icon, and selecting the task required from a menu. The system can be operated with a touch screen and a numeric keypad, providing a friendly and intuitive interface to the user. This mode of operation makes the integration of the computer hardware onto the relatively hostile environment of the shop floor more feasible.

Figure 6 illustrates the relationship of the user interface to the agent, rules, and the object hierarchy. When the user issues a command to the expert agent, the expert interprets this command and selects the appropriate message, or messages to send to the inference engine to initiate the interaction of relevant rule sets with the object hierarchy. The rule sets have been prioritised to facilitate cost effectiveness when diagnosing faults. For example, given that the expert agent may have two rule sets available to determine the probable cause of a fault indication, it will use the set of rules which should pinpoint the solution in the shortest time. The results of the inference process are sent back to the user, through the expert agent. It decides what resources (i.e. in textual, pictorial, or graph form) are relevant to best highlight the diagnosis, and sends that information to the screen.

In addition to using the expert as agent metaphor, the user can browse through the domain objects of the on-screen data sheet, and access technical references at each stage.



EEC Checks

	PIN	PARA 11.3.19	
Power Lever	36	-4.099	V
Torque	1	0.299	V
Delta P/P	13	-1.575	V
TT2	7 + 8	0.439	V

	PIN	PARA 11.3.19	
Temperature Limiter	21	3.0	V
Torque Limiter	37	9.6	V
Delta T	12	-2.435	V
EGT	22	-0.010	mV
Temp Set Point	20	0.003	mV

Information and Diagnosis

Limiter check: one of the pins is giving a reading
00V & pin 37: 9.60V
k power
ading in pin 37 is 9.60 V.
00% +- 2 .
Either the gauge or the EEC is at fault...
I'll compare the gauge reading to the calculated torque (from pin 1)
Gauge reading: 95.00%, calculated torque reading: 0.00%

It's not the gauge or the wiring, so it must be the EEC that's at fault

Figure 5. User Interface.

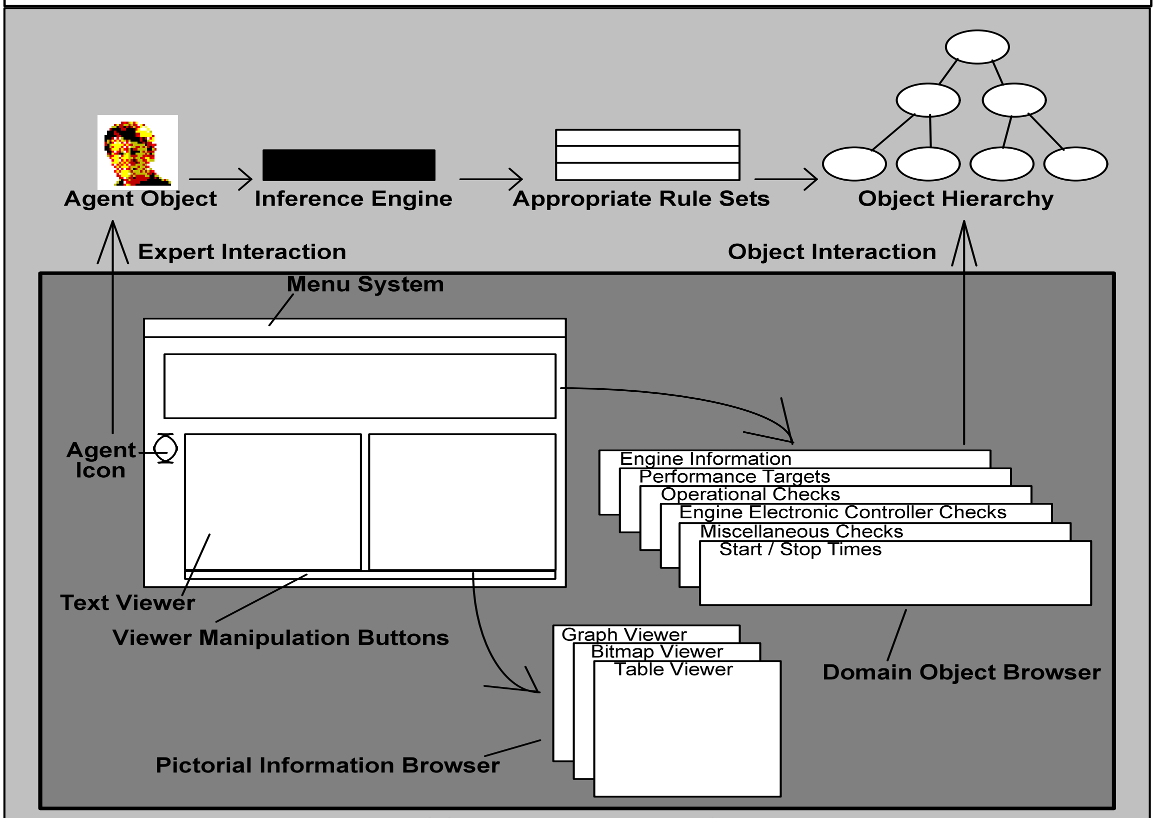


Figure 6. Layout of Screen in relation to System Architecture

5 CONCLUSIONS

The simplification of the AOP paradigm into the expert as agent metaphor provides an uncomplicated addition to the KBS tool kit, which, when combined with information stored as objects, begins to address the problem of integrating knowledge bases with object-based information.

Most important, however, is the resulting **usability** of such systems. The expert metaphor is at its most helpful in the user interface. The user can browse through objects, but the main load of interaction is carried out through the expert. Thus the system mimics the real world, where the expert is consulted to solve a problem. In addition, the expert can be a peer entity to the user, which provides an exciting method of modelling the user through stereotypes. For example, if the user is an unskilled operator, the agent on-screen will appear as an operator expert, whilst a manager using the same system would interact with a manager expert agent.

In conclusion, systems that combine OOP and KBS using the expert as agent metaphor, provide a *knowledge browsing* facility where the user can access textual and pictorial information. This combination of Computer-Assisted Learning (CAL) and KBS assists in technology transfer, and gives the user a flexible learning environment, where skills *and* CAL-type material coexist.

References

Graham91	Graham, 1991. Ian. Object-Oriented Methods, Addison-Wesley
Minsky75	Minsky, M. A. 1975. Framework for Representing Knowledge. In: The Psychology of Computer Vision. Winston, P. (ed.) McGraw-Hill.
Torsun89	Torsun, I. S. & Ng, Y. M. 1989. Tightly-Coupled Expert Database Systems Interface In: Research and Development in Expert Systems V. Proc. Expert Systems 1988. Kelly, B & Rector, A (eds) Cambridge.
Shoham93	Shoham, Yoav. 1993. Agent-oriented Programming. Artificial Intelligence. 60, 51-92.
Hewitt85	Hewitt, C. E. 1985. Viewing Control Structures as Patterns of Passing Messages. J Artificial Intelligence. 8, 3.
Nii86	Nii, H. P. 1986. Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. AI Magazine 7, 2.
Wielinga92	Wielinga, B. J., Schreiber, A. T. and Breuker, J. A. 1992. KADS: A Modelling Approach to Knowledge Engineering. Knowledge Acquisition. 4, 5-53.
Kingston92	Kingston, J. 1992. Pragmatic KADS: A Methodological Approach to a Small Knowledge-Based Systems Project. Expert Systems. 9, 4.