OBJECT-ORIENTED IMPLEMENTATION OF OBJECT-ORIENTED KNOWLEDGE REPRESENTATION

B R Gaines
Knowledge Science Institute, University of Calgary
Calgary, Alberta, Canada T2N 1N4
(gaines@cpsc.ucalgary.ca, (403) 220-5901)

The design and application of KRS, an object-oriented knowledge representation server in the KL-ONE, KRYPTON, CLASSIC family, are described. The implementation of KRS as a class library in an object-oriented language using generic, reusable objects is analyzed. The graphic editing tools developed in the object-oriented language to support ease of interactive editing of the objects in the knowledge representation system are described and illustrated. It is concluded that the object-oriented paradigm is highly effective at both the representation and implementation levels.

INTRODUCTION

The practical application of knowledge representation and reasoning systems presents a major opportunity for empirical research on fundamental issues of artificial intelligence as well as practical applications. The utopian objective is the empirical study of significant applications of highly principled knowledge-based systems. This involves systems designed in such a way that maximum achievable performance is delivered through a transparent implementation that verifiably exhibits the principles under investigation. The problems of achieving this combination of significant application, deep principles and proper software engineering are not unique to artificial intelligence research—for example, operating systems, databases and real-time systems have all had similar needs for, and limitations on, empirical studies.

There have been two recent developments that open up new possibilities for meaningful empirical research in artificial intelligence: new insights into the complexity issues affecting knowledge representation and reasoning (Brachman & Levesque, 1984; Nebel, 1990); and the development of object-oriented programming environments offering precise definition and efficient implementation of extensible abstract data types (Shriver & Wegner, 1987). The advances in artificial intelligence allow requirements specifications to be developed for knowledge representation and reasoning systems with known, and practically applicable, performance curves (MacGregor, 1988; Borgida, Brachman, McGuinness & Resnick, 1989). The advances in software engineering allow these specifications to be implemented cleanly in such a way that their conformance with the requirements can be verified and their algorithms achieve the best possible performance.

The application requirement that motivated the development of the knowledge representation server reported in this paper was that of *knowledge acquisition*, from human experts and through induction from datasets, for the development of knowledge-based systems. Acquisition has long been recognized as a major bottleneck in expert system development (Hayes-Roth, Waterman and Lenat, 1983), and research on knowledge acquisition has developed a massive literature in recent years (Boose & Gaines, 1988, 1990; Gaines & Boose, 1988; Marcus, 1988). We have developed knowledge acquisition toolkits combining graphic knowledge editing, repertory grid elicitation, conceptual clustering and empirical induction (Gaines, 1988; Gaines & Shaw 1989, 1990) that have been widely used both in other research groups and, industrially, as commercial products (Rappaport & Gaines, 1988; Gaines & Linster, 1990). The generality and range of applications of such toolkits has been limited by the representation and reasoning schema used (that of entity-attribute databases with categorical data types and production rules). Problems have also arisen in managing the complexity of integrating many different techniques within the toolkits (Boose & Bradshaw, 1987).

The system reported in this paper is the result of a complete redesign of these knowledge acquisition toolkits as a set of principled modules communicating with, and through, a very general knowledge representation server. The modules and the server have been implemented as a class library in C++, both for practical reasons of portability and speed, and also for the potential to develop a simple architecture based on the use of a few reusable classes whose operation is very clear and can be easily verified. A number of the new acquisition tools have now been used on a range of projects, and by other developers, for applications including the rational reconstruction of some large-scale knowledge-based systems.

KNOWLEDGE REPRESENTATION SERVER REQUIREMENTS

Figure 1 shows the architecture of KRS. It is designed to manage multiple standard knowledge bases and to interface to other servers either of the same or differing architectures.

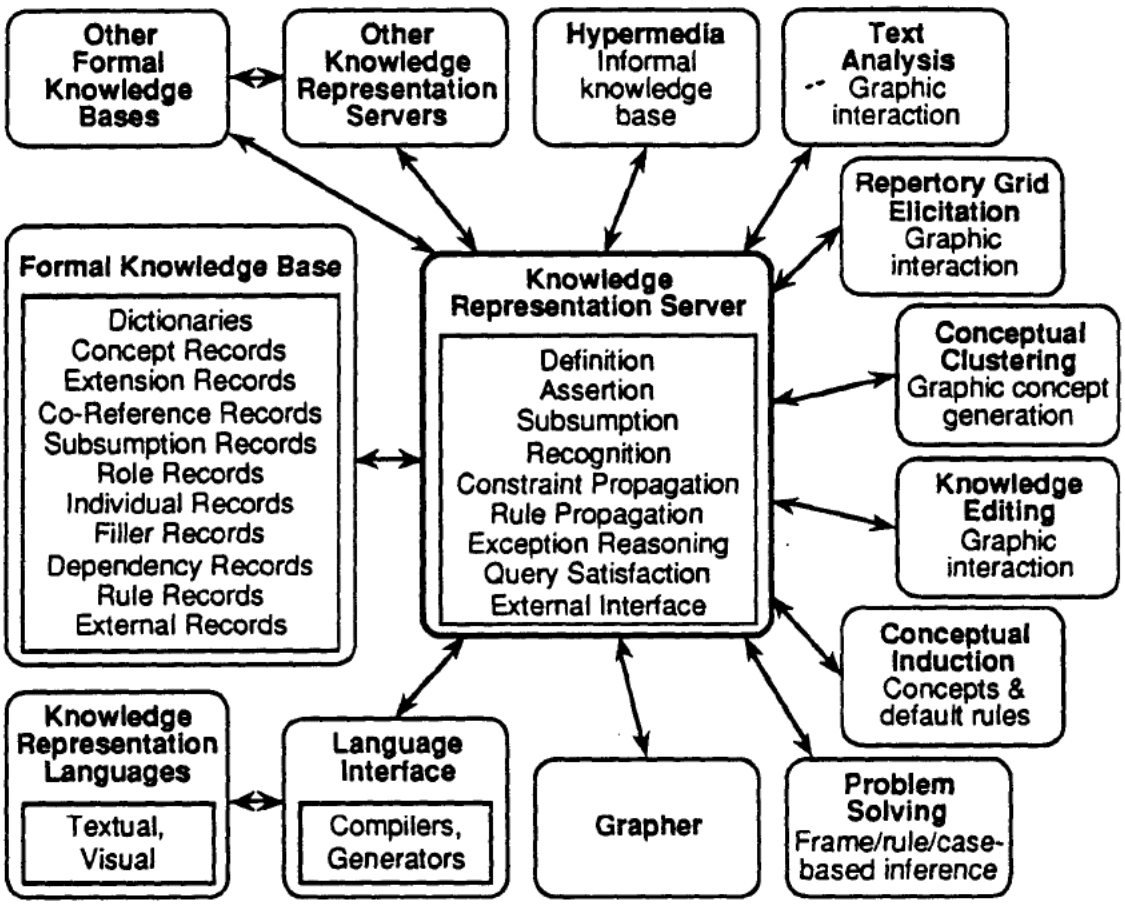

Fig.1 Architecture of the knowledge representation server

The various sub-modules attached to the server are options offering specific additional functionality. The modules are (reading clockwise from the top left):

- Interface modules to other formal knowledge bases and knowledge representation servers including databases.
- A hypermedia module allowing informal knowledge structures in text and images to be captured, accessed and linked. The linkage structure itself is held as a knowledge base.
- A text analysis module allowing documents to be analyzed in terms of word usage, and associations between significant words to be graphed—based on TEXAN in KSS0 (Gaines 1988). This enables protocols and technical documents to be used to initiate knowledge acquisition.

The system reported in this paper is the result of a complete redesign of these knowledge acquisition toolkits as a set of principled modules communicating with, and through, a very general knowledge representation server. The modules and the server have been implemented as a class library in C++, both for practical reasons of portability and speed, and also for the potential to develop a simple architecture based on the use of a few reusable classes whose operation is very clear and can be easily verified. A number of the new acquisition tools have now been used on a range of projects, and by other developers, for applications including the rational reconstruction of some large-scale knowledge-based systems.

KNOWLEDGE REPRESENTATION SERVER REQUIREMENTS

Figure 1 shows the architecture of KRS. It is designed to manage multiple standard knowledge bases and to interface to other servers either of the same or differing architectures.

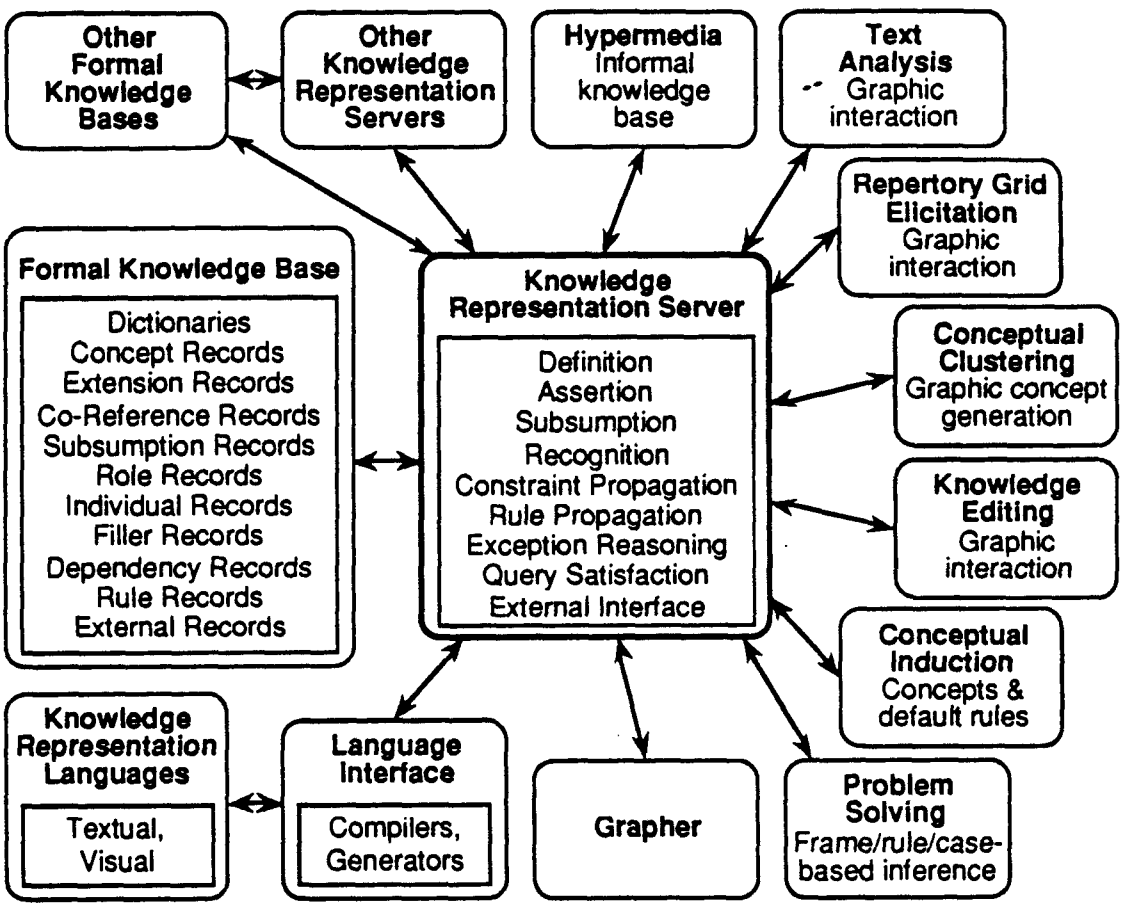

Fig.1 Architecture of the knowledge representation server

The various sub-modules attached to the server are options offering specific additional functionality. The modules are (reading clockwise from the top left):

- Interface modules to other formal knowledge bases and knowledge representation servers, including databases.
- A hypermedia module allowing informal knowledge structures in text and images to be captured, accessed and linked. The linkage structure itself is held as a knowledge base.
- A text analysis module allowing documents to be analyzed in terms of word usage, and associations between significant words to be graphed—based on TEXAN in KSS0 (Gaines, 1988). This enables protocols and technical documents to be used to initiate knowledge acquisition.

- A repertory grid expertise transfer module allowing graphic definition of concepts and graphic creation and editing of individuals—based on the elicitation screens of KSS0 (Gaines & Shaw, 1990).
- A conceptual clustering module allowing interactive definition of new concepts—based on the hierarchical and spatial clustering from KSS0.
- A knowledge editing module allowing the interactive development and editing of knowledge structures through a visual language.
- A conceptual induction module creating rules about specified subsets of individuals and transforming them to a minimal set of concepts and default rules—based on the Induct algorithm (Gaines, 1989).
- A problem solving module supporting frame, rule and case-based inference from the knowledge structures.
- A grapher laying out specified parts of the concept subsumption graph, concept structures and individual structures—based on an incremental layout algorithm by Watanabe (1989) that can be used interactively to support the production of clear visual knowledge structures.
- A language interface accepting and generating definitions and assertions in formal knowledge representation languages, both textual and visual.

The language interface supports a table-driven top-down recursive-descent compiler for CLASSIC-like languages. The tables can be changed dynamically in a prelude to the compiler allowing a syntax that makes clear distinctions between definitions and assertions to be used in examples, and a less verbose language to be used for system development. The same tables are used to output definitions and assertions in the language so that graphically entered concepts and individuals may be described, stored and ported to other systems in a textual form. Like all the peripheral modules, the language interface module is easily changed for another, and there is no implied commitment to the current language.

The language interface also supports visual knowledge representation languages similar to those of the graphic knowledge editor but allowing offline drawings to be parsed as formal specifications. This is useful for standard ontologies which may be presented effectively using drawing tools but where the drawing themselves are also the formal specification.

DATA STRUCTURES

It was proposed that a class library in an object-oriented language has the potential to provide a simple implementation based on a few, verifiably principled modules. Figure 2 shows the objects used in the server implementation and their data structures. The basic class library provides variable-length record objects which are sub-classed to provide dictionaries, sets and record objects for concept and individual records. The objects are:

- *Concept records* hold the constraints defining a concept: its primitives and those disjoint from them; its extensions as the subsets of individuals that must be, and may be, included in any subset recognized; its cardinality and co-reference constraints on any subset recognized; its rule and release concepts, if any, which will be asserted of any individual recognized, or impossible to recognize, respectively; and, for each role constrained in the concept, the concept constraining it.
- *Individual records* hold the templet concept defining intensional constraints upon an individual and, for each role constrained in the templet or filled in the individual, a specification of the role fillers.
- *Filler records* hold subsets of individuals that fill slots.
- *Dictionaries* recognize concept, role and individual names and generate an accession number used to index other structures. Syntactically there is a clear separation between concepts, roles and individuals and a lexical item can occur in more than one dictionary with different meanings.

- *Extension records* hold subsets of individuals, typically those that may occur in a role filling set and those that must occur in it.
- *Data records* hold subsets of external individuals that are explicitly recognized by the server, notably integers, reals, dates and strings.
- *Co-reference records* hold equivalence subsets of role chains expressing co-reference constraints.
- *Subsumption records* hold computed subsumption relations on a three-valued basis, that one concept subsumes another, does not, or cannot.
- *Dependency records* keep track of inverse relations between individuals, supporting explicit inverse roles, efficient rule propagation and efficient retraction.
- *External records* hold tables translating between internal and external concepts and individuals.

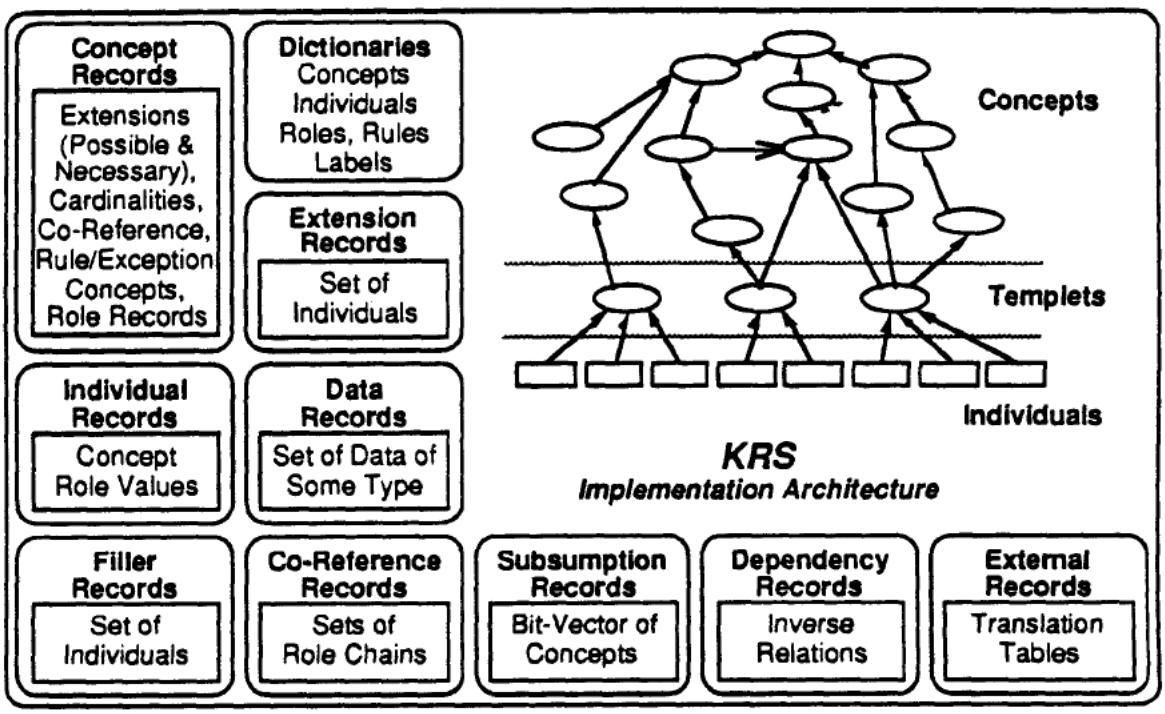

Fig.2 Data structures in the knowledge representation server

The server supports the knowledge representation facilities defined for CLASSIC (Borgida, Brachman, McGuinness & Resnick, 1989), with the following extensions:

- Automatic filling of *inverse roles*.
- *Negative set definitions* can be used for extensions, for example, that a role cannot be filled by red or blue. This is technically simple, causes no increase in complexity, and leaves subsumption well-defined, but substantially increases representation power, particularly in applications where the open-world semantics are significant.
- *Integer, real, date* and *label* (string) data types are fully supported and the accompanying range constraints are incorporated in subsumption computation.
- *Rules* are named objects supporting exceptions.
- Subsumption and recognition are *three-valued*, true, false and open.

The implementation separates facts, individual conceptual constraints, and inferences. This makes rule propagation and retraction fast and simple, and enables light-weight 'possible worlds' to be supported. It is also very significant in the knowledge acquisition context in enabling the sources of responses to queries to be shown clearly to users.

A number of optimizations are applied in the implementation to minimize storage and maximize speed. For example, special codes are used for the empty, universal and singleton sets such that the set objects never store them, and subsumption is calculated in such a way that only a fast lookup is needed to check subsumption between role concepts rather than a recursive call. One advantage of the object implementation is that these optimizations can be put in override methods in sub-classes so that the kernel implementation is small and easily checked but potentially slow, and the full implementation can be checked against the kernel for correctness and optimized for speed.

Figure 3 shows the complete class library as an extension of Think C's standard class library on the Macintosh. The classes written for KRS are shown with bold rounded boxes.

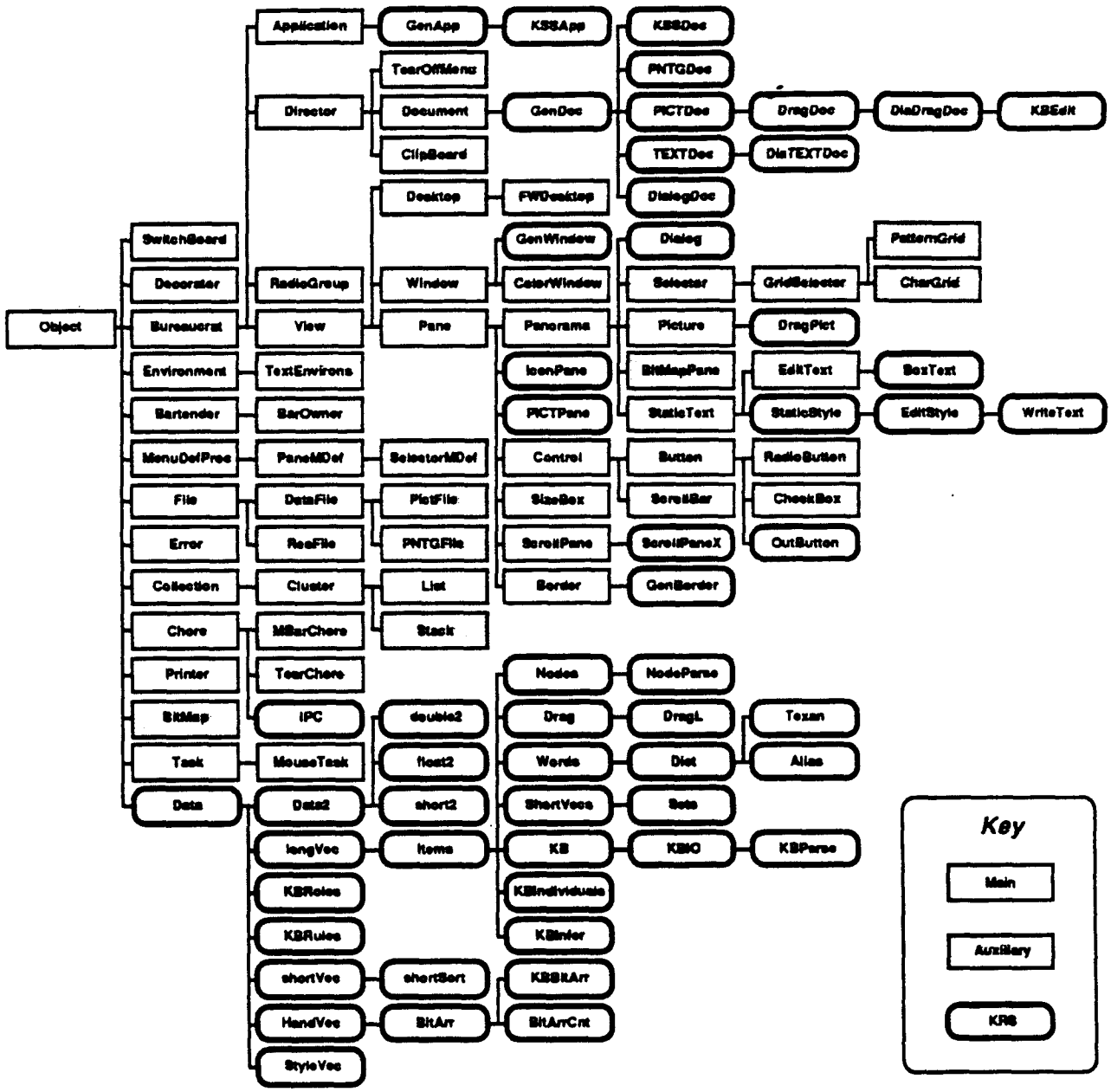

Fig.3 The complete class library for KRS

The only significant class in the top half of the diagram is **KBEdit** at the top right. It implements KRS's graphic knowledge class/object editor described in the next section. KBEdit is at the end of a long inheritance chain:

Object—the Think C top level class of objects definition
Bureaucrat—a class for objects that manage other objects
Director—a class for objects that manage a window
Document—for managing a file and associated window
GenDoc—managing a range of generic files and associated window
PICTDoc—managing a graphic file and associated window
DragDoc—managing an interactive graphic drawing file/window
DiaDragDoc—managing interactive graphic drawing with associated controls
KBEDIT—specializes this for graphic editing of KRS knowledge structures

DragDoc uses component objects inheriting from **DragPict,** a class supporting graphic drawing and interaction, which in turn uses components inheriting from **DragL**, a class supporting data structures for graphic objects with connecting lines and arrows.

The main knowledge base structures are in the lower part of Figure 3 where **KBParse** at the lower right is a KRS knowledge base with the inheritance chain:

Object—the Think C top level class of objects definition
Data—a class for variable length data structures
longVec—variable length vectors of longs
Items—variable length vectors of variable length items
KB—variable length vectors of KRS class definitions
KBIO—adding input/output functionality
KBParse—adding language parsing functionality

KBParse uses component objects inheriting from **Items** such as:

KBIndividuals—KRS object data
KBInfer—dependencies between KRS objects (for rapid retraction)
Sets—set data records as shown in Figure 2
Alias—dictionaries of names as shown in Figure 2

It also uses component objects inheriting from **Data** such as:

KBRoles—inverse roles and role chains
KBRules—rule data structures
KBBitArr—subsumption tables (for rapid recognition of objects in classes)

Other relevant classes in Figure 3 are:

Texan—implementing the text analysis and clustering module of Figure 1
NodeParse—implementing Watanabe's graph layout facilities
IPC—implementing an inter-application communication protocol used to communicate with expert system shells and hypermedia systems

The most important conclusion that can be drawn from Figure 3 is the relative paucity of classes that are specific to KRS, namely **KSSApp, KSSDoc, KBEdit, Texan, KB, KBIO, KBParse, KBIndividuals, KBInfer, KBRoles, KBRules, KBBitArr**. Other classes are highly generic to any interactive graphic application or to general data processing, and we are using them in a range of applications such as hypermedia and groupware. The **BitArr** class will be heavily reused in our transfer of the Induct induction algorithm from KSS0 to this environment. The **DragPict** class was designed to support the interactive click and drag repertory grid knowledge elicitation techniques developed for KSS0 and KITTEN and will again be heavily reused as they are transferred.

KRS GRAPHIC CLASS/OBJECT EDITOR

Classes and objects in KL-ONE-style knowledge structures are difficult to edit because they require the 'top-down' definition of general concepts and their stepwise refinement to more specific concepts. This is reasonable in well-established domains where the ontology has been developed as part of a professional discipline, but it is rarely a suitable basis for knowledge acquisition in novel domains. Experts often develop their knowledge structures 'bottom-up' from cases through commonality to general ontologies, and find a 'top-down' approach unnatural. There are a number of ways of overcoming this problem—the conceptual clustering and induction tools develop concept hierarchies 'bottom-up' from cases—but general concept editing services are difficult to define.

The problem is to design an interface allowing the person editing to adequately express his or her intentions. For example, if a concept is edited it is simple enough to maintain referential integrity by disallowing use of concepts defined after it in the definition, and even this constraint can be relaxed in a more intelligent editor that places a new or edited concept in an appropriate place in the referential order if one exists. However, it is difficult to determine whether the user intends, when editing, that subsumption relations be preserved and hence that edits should be propagated, and, if so, whether the user really wants full propagation or the option of checking what is occurring.

The BLIP knowledge acquisition tool developed at the Technical University of Berlin to support 'sloppy modeling' (Morik, 1987) provides a conceptual system development and editing tool that computes all consequences of definitions dynamically, including induction from cases. For the initial stages of knowledge acquisition even this may be too strong in that experts often generate inconsistent knowledge structures but are not ready to resolve the inconsistencies until they have a developed a fairly large-scale model giving them an overall perspective on the task and domain.

The KRS graphic knowledge editing tool shown on the center right of Figure 1 supports this relaxed approach. Knowledge structures can be developed and edited using drawing tools providing a natural visual language. The structures can extend across multiple screens supporting modularity and ontology 'libraries.' Structures can be freely duplicated and definitions may be fragmented across screens. The consistency of the definitions is determined only when they are compiled into a formal knowledge structure and inconsistencies are graphically highlighted.

Figure 4 shows a screen from the graphic knowledge editor on which the minimal five rule default reasoning solution of Cendrowska's (1987) contact lens prescription problem has been developed. This screen compiles directly into a set of assertions and rules in KRS as shown in Figure 5. The top diagram on the screen defines the concept of a patient with possible attributes and values. The center diagram defines the concepts of a lens recommendation and three potential recommendation. The lower diagram defines the default rules.

The grapher is highly interactive. Objects, and groups of objects, may be dragged to new locations and the lines remain connected. Objects may be double-clicked to select them for editing—both text and object type may be edited. Popup menus associated with each object (see upper right of Figure 4) enable the connecting lines and arrows to be entered simply.

The visual language used in Figure 4 is precisely defined. Concepts are ovals, primitive concepts are ovals with small horizontal lines inside each side, individuals rectangles, roles (and annotation) unboxed text, rules rounded-corner boxes, constraint expressions rounded-corner boxes with small horizontal lines. The interpretation of the arrows is overloaded but well-defined by the types of the objects at their head and tail. Lines without arrows connecting primitive concepts denote that the concepts are disjoint, and connecting roles that they are inverse.

The compiler traces concepts and individuals through their outgoing arrows, builds a dependency table for concepts, highlighting circular links that it will not use, and generates concept definitions and individual assertions.

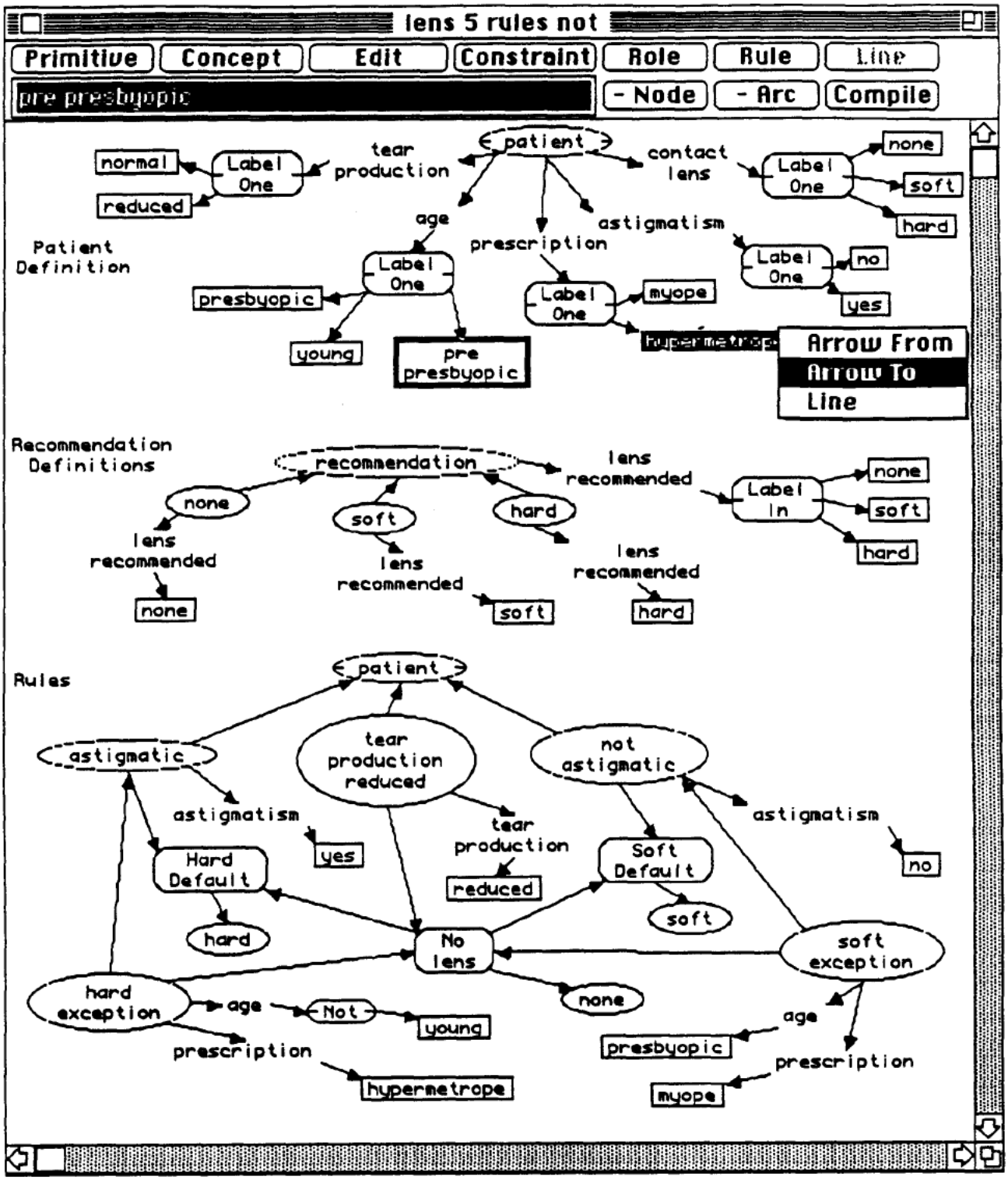

Fig.4 Graphic knowledge editor

```
Primitive(patient,
  (All tear production, (Label One normal, reduced))
  (All astigmatism, (Label One yes, no))
  (All contact lens, (Label One none, hard, soft))
  (All age, (Label One young, presbyopic, pre presbyopic))
  (All prescription, (Label One myope, hypermetrope))
)
Concept(recommendation,
  (All lens recommended, (Label In none, hard, soft))
)
Concept(none, recommendation,
  (All lens recommended, (Include none))
)
Concept(soft, recommendation,
  (All lens recommended, (Include soft))
)
Concept(hard, recommendation,
  (All lens recommended, (Include hard))
)
Concept(astigmatic, patient,
  (All astigmatism, (Include yes))
  (Then Hard Default)
)
Concept(not astigmatic, patient,
  (All astigmatism, (Include no))
  (Then Soft Default)
)
Concept(tear production reduced, patient,
  (All tear production, (Include reduced))
  (Then No lens)
)
Concept(hard exception, astigmatic,
  (All prescription, (Include hypermetrope))
  (All age, (Not young))
  (Then No lens)
)
Concept(soft exception, not astigmatic,
  (All prescription, (Include myope))
  (All age, (Include presbyopic))
  (Then No lens)
)
If(Hard Default, hard)
If(Soft Default, soft)

If(No lens, none)
Unless(No lens, hard, soft)
```

Fig.5 Resultant knowledge base compiled

INTEGRATION

An obvious criterion against which to test a server is its ease of integration with other systems. KRS is accessible at three levels, as an extensible C++ class library, as an embedded, callable module, and as a server that may be accessed through messages using an inter-process communication protocol which also operates across a network. Our own applications involve the development of the class library, but there have also been a number of applications by other system developers that use KRS strictly as a server.

Woodward's (1990) Cognosys application is a knowledge acquisition tool designed to analyze free text forms of expert protocols and interviews in terms of Graesser and Clark's (1985) *implicit knowledge structures*. These involve four types of knowledge object among some combinations of which there can be up to eight kinds of relations. Cognosys provides a wide variety of graphic interactive tools for the analysis of the data structures but did not have the means for checking, applying and displaying them.

A module was written within Cognosys in which the Graesser and Clark structures were represented as concepts in KRS's language, and the data structures in terms of them were represented as individuals. Inter-process communication was used to transmit these to the KRS server together with user-generated requests to graph individuals satisfying certain relations, or the conceptual structure, or errors in its application determined by type-checking rules written for KRS. Reverse communication from the server to Cognosys allows interaction with the individuals in the KRS grapher to retrieve the original text protocols in Cognosys.

Figure 6 shows a version of the Cognosys query generator for interaction with KRS in a server mode, including the original data structures and their translation to KRS intermediate language. Figure 7 shows part of a knowledge structure graphed in KRS in response to a Cognosys query. The graph layout here is fully automatic, designed to maximize the use of space and minimize line crossings.

Similar interfaces have been written for Shaw's repertory grid knowledge acquisition tools (Shaw & Gaines, 1987) where the concepts and cases elicited and rules induced are communicated as a knowledge structure to KRS and may be graphed or used in inference.

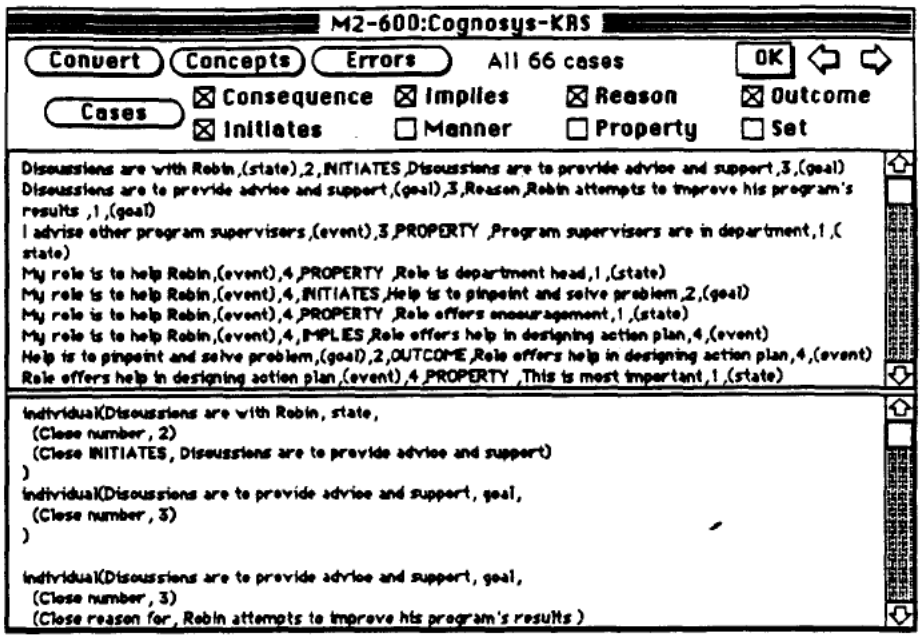

Fig.6 Cognosys data and query generator for calls to KRS server

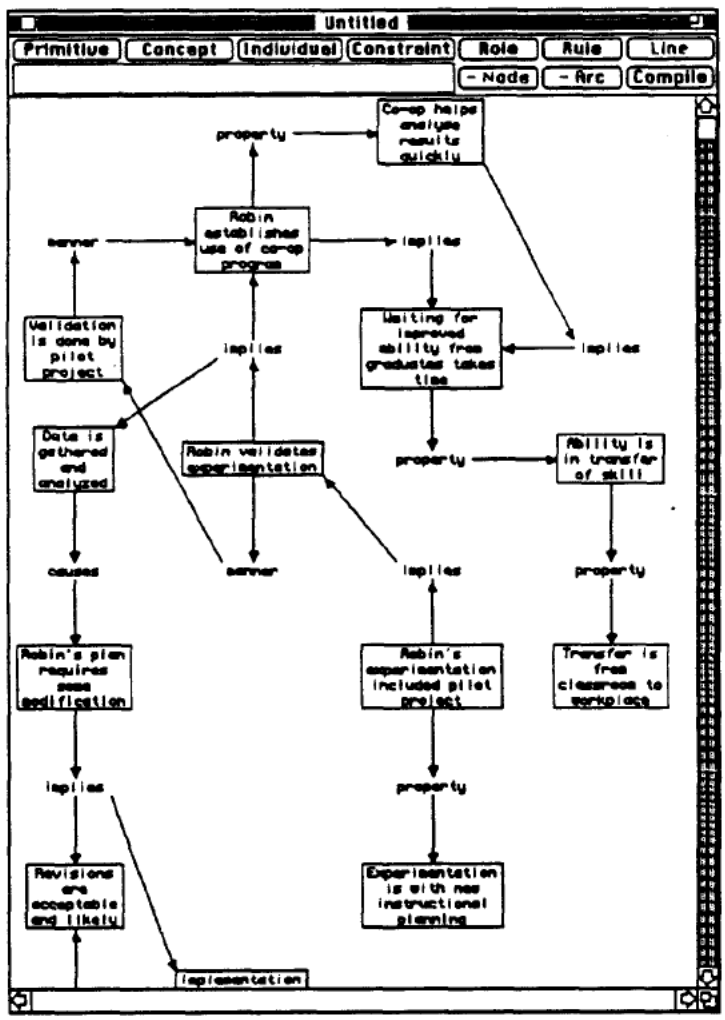

Fig.7 Some data displayed as a result of a Cognosys query

EXPERIENCE IN DIFFERENT PROBLEMS DOMAINS

The C++ implementation of KRS was designed to be efficient in space and fast in speed. This has been evaluated on some large scale problems. For example, the Garvan Institute of Medical Research thyroid diagnosis knowledge base, GARVAN-ES1, has been in use for many years, covering some 60 diagnoses through some 650 rules and having associated case datasets of over 20,000 patients. It provides a realistic test case for 'expert system' applications and is of particular interest as it is being re-analyzed by a number of researchers using different techniques. In particular, Compton and Jansen (1990) have used it to examine the improvements in knowledge acquisition and maintenance possible with their 'ripple-down rules' representation based on a subsumption hierarchy with dependencies between rules at the same level. The KRS rule system for reasoning with exceptions fully supports the ripple-down technique—the rule set becomes a single chain in which the first rule to fire acts as an exception to all others.

The ripple-down version of the Garvan rules, 547 rules, together with 669 cases evaluated on 17 attributes, were loaded and run in KRS on a Macintosh II. The total number of concepts generated was 1035, and the space occupancy was 315Kbytes, of which 86K was for concepts, 141K for the subsumption bit-map and 58K for the individual cases. The diagnoses took 0.4 seconds each on average.

As another example, to encourage comparative evaluation of different tools on the same problem, the European Knowledge Acquisition Workshop in 1990 instigated Project Sisyphus in which a collection of knowledge-based system development tasks are being circulated to tool developers and users. We have used KRS with a room configuration task submitted from the GMD in Germany which is being worked on there and at other sites under an ESPRIT project (Voß, Karbach, Drouven, Lorek & Schuckey, 1990).

The problem is one of allocating individuals to rooms under a set of, possibly conflicting, constraints, many of which involve relations between types of individual. The data set consists of sample cases and protocols from an expert solving some of them. The problem is an interesting one for a term subsumption knowledge representation system since it goes well beyond simple classification and involves some kind of planning under constraints.

The KRS grapher proved a very effective tool in entering the basic knowledge structures since the organization chart, building layout, and so on, have a natural graphical representation which it was easy to mimic in the grapher. This made validation of the knowledge structures particularly simple. Figure 8 shows the conceptual structures at the top, facts about a sample case in the center, and some of the problem-solving strategies at the bottom.

It was not clear at the start of this project how much contribution KRS could make to the problem solving component. The GMD group had attached a constraint satisfaction engine to a KL-ONE-like knowledge representation system in their solution. It was remarkably easy, however, to program most of the solution within KRS without additional features in a number of different ways. The cleanest solution seems to be a blackboard, or agenda, mechanism, in which the classification of people causes requests to be placed on an agenda which are satisfied by the classification of rooms. The symmetry of this consumer-supplier arrangement is particularly attractive in dealing with peculiarities where the logic reverses, such as finding another smoker to fill a room to which a smoker has already been allocated.

The agenda mechanism was programmed as a set of concepts, individuals and rules in KRS. The only external component involved in the solution is a simple selection mechanism that accepts solutions placed on the agenda and implements them by asserting that a person is allocated to a room. This needs an external mechanism to choose between alternative solutions. However, this choice involves no 'intelligence', only the capability to make a succession of arbitrary choices and to backtrack by retraction should a sequence lead to an impasse in which no further solutions are possible, that is, a simple search procedure.

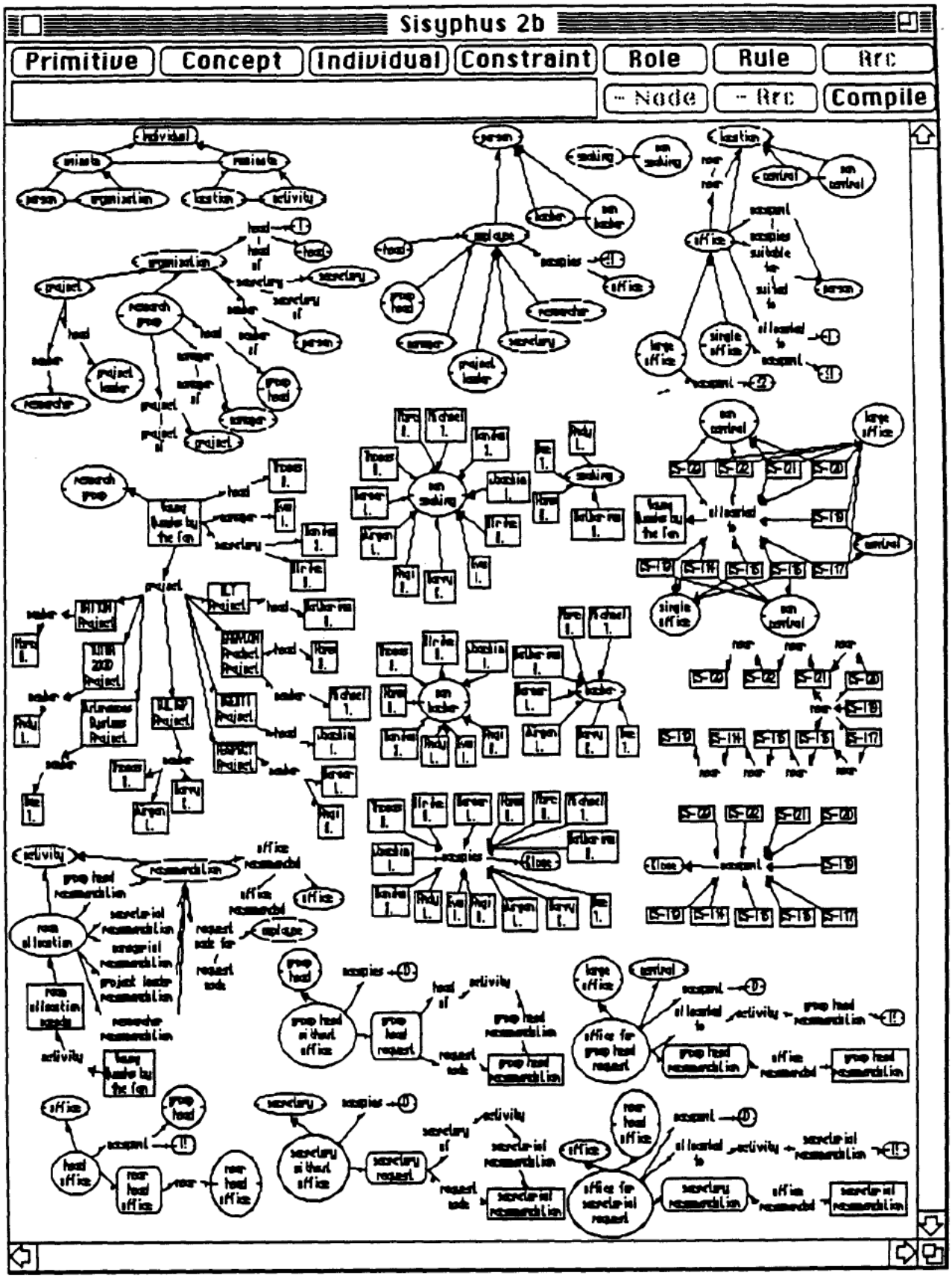

Fig.8 Room allocation knowledge structures

CONCLUSIONS

The main objective of this paper has been to demonstrate that open architecture, object-oriented knowledge representation servers are straightforward to implement using an object-oriented language, and offer opportunities for the empirical investigation of principled artificial intelligence in practical application. The implementation of the server with the functionality and optimizations described is fast and storage efficient, and was easily implemented. We have run it with knowledge bases of several thousand concepts and individuals from a variety of sources including the GARVAN-ES1 described above, our sports knowledge bases (Gaines & Vickers, 1988), and a variety of artificial datasets designed to stress the code.

Keeping subsumption up-to-date appears to be a negligible overhead even with very large numbers of concepts. Recognition has not been a problem with the knowledge acquisition datasets that tend to consist of numbers of self-contained knowledge islands. However, it is possible to define knowledge structures that involve substantial propagation over large sets of individuals each time a new assertion is made and this would cause interactivity problems if it was a significant application requirement. This is a domain problem rather than one of server implementation and it would be reasonable to expect it if the system were used to investigate, for example, the mathematics of algebraic structures. Similar problems would arise if configurational problems involving many interacting constraints were represented for solution in the system.

The implications of accepting the restrictions necessary to make subsumption and recognition tractable in the server have still to be investigated in the context of knowledge acquisition. So far, KRS provides such a major increase in representation capabilities compared with previous knowledge acquisition systems that it has not been a major issue. Our design philosophy is that the analogy to arithmetic units holds, and it is appropriate to develop clean, complete and fast units with limited capabilities, and then supplement them with other units providing additional functionality ranging from general theorem provers to highly domain-specific modules. This is particularly important in a knowledge acquisition environment where one has to present a clean conceptual model and user interface to users who may never have used a computer before and are attempting to develop overt knowledge structures where none previously existed.

Finally, as already noted in the literature (De Troyer, Keustermans & Meersman, 1986), the object-oriented implementation contributes nothing directly to the object-oriented system being implemented. That is, there is no trace of the C++ classes in the KRS knowledge structures. What is contributed, as noted in the literature comparing C++ and Lisp (Trickey, 1988), is a superb software engineering environment supporting extremely well-structured and transparent implementation of conceptually difficult artificial intelligence knowledge representation and inference processes. The two levels of object-oriented system in KRS are conceptually independent but the lower level is pragmatically highly important to the design and implementation of the upper level.

This difference in levels shows up most clearly in issues such as multiple inheritance. We have found no need for multiple inheritance in the C++ implementation language. However, multiple inheritance is a basic requirement and an essential feature of KRS. A simple explanation of this difference is that at the implementation level one has full conceptual control and it easy to incorporate the effects of multiple inheritance through appropriate structures in a single inheritance system. At the knowledge level, however, one is attempting to represent cognitive structures without distortion and multiple inheritance is common in human conceptual frameworks.

ACKNOWLEDGEMENTS

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada. I am grateful to Paul Compton for access to the Garvan ES-1, to Marc Linster for the Sisyphus data set, and to Mildred Shaw for collaboration in research and comments on this paper.

REFERENCES

Boose, J.H. & Bradshaw, J.M. (1987). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **Int. Journal of Man-Machine Studies, 26,** 3-28.

Boose, J.H. & Gaines, B.R., Eds. (1988) **Knowledge Acquisition Tools for Expert Systems.** London, Academic Press.

Boose, J.H. & Gaines, B.R., Eds. (1990) **The Foundations of Knowledge Acquisition.** London, Academic Press.

Borgida, A., Brachman, R.J., McGuiness, D.L. & Resnick, L.A. (1989) CLASSIC: a structural data model for objects. Clifford, J., Lindsay, B. & Maier, D., Eds. **Proceedings of 1989 ACM SIGMOD International Conference on the Management of Data.** pp.58-67. New York: ACM Press.

Brachman, R. J., & Levesque, H. J. (1984) The tractability of subsumption in frame-based description languages. **Proceedings of AAAI-84.** pp.34-37. San Mateo, California: Morgan Kaufmann.

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

Compton, P. & Jansen, R. (1990) A philosophical basis for knowledge acquisition. **Knowledge Acquisition 2**(3).241-258.

De Troyer, O., Keustermans, J. & Meersman, R. (1986) How helpful is an object-oriented language for an object-oriented database model? **Proceedings of 1986 International Workshop on Object-Oriented Database Systems.** pp.124-132. Washington: IEEE Computer Society Press.

Gaines, B.R. (1988) Knowledge acquisition systems for rapid prototyping of expert systems. **INFOR, 26**(4), 256-285 (November).

Gaines, B.R. (1989) An ounce of knowledge is worth a ton of data: quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. **Proceedings of the Sixth International Workshop on Machine Learning.** pp.156-159. San Mateo: Morgan Kaufmann (June).

Gaines, B.R. & Boose, J.H., Eds. (1988) **Knowledge Acquisition for Knowledge-Based Systems.** London, Academic Press.

Gaines, B.R. & Linster, M. (1990). Integrating a knowledge acquisition tool, an expert system shell and a hypermedia system. **International Journal of Expert Systems Research and Applications** to appear.

Gaines, B.R. & Shaw, M.L.G. (1989) Comparing the conceptual systems of experts. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.** pp.633-638. Los Angeles, California: Morgan Kaufmann (August).

Gaines, B.R. & Shaw, M.L.G. (1990). Eliciting knowledge and transferring it effectively to a knowledge-based systems. **IEEE Transactions on Knowledge and Data Engineering** to appear.

Gaines, B.R. & Vickers, J.N. (1988) Design considerations for hypermedia systems. **Microcomputers for Information Management,** 5(1), 1-27 (March).

Graesser A.C. & Clark, L.F. (1985). **Structures and Procedures of Implicit Knowledge.** New Jersey: Ablex.

Hayes-Roth, F., Waterman, D.A. & Lenat, D.B., Eds. (1983) **Building Expert Systems.** Reading, Massachusetts: Addison-Wesley.

MacGregor, R.M. (1988) A deductive pattern matcher. **Proceedings of AAAI88**. pp.403-408. San Mateo: Morgan Kaufmann.
Marcus, S. (Ed.) (1988) **Automating Knowledge Acquisition for Expert Systems**. New York: Kluwer.
Morik, K. Ed. (1985) **Knowledge Representation and Organization in Machine Learning**. Berlin: Springer-Verlag.
Nebel, B. (1990) **Reasoning and Revision in Hybrid Representation Systems**. Berlin: Springer-Verlag.
Rappaport, A. & Gaines, B.R. (1988). Integrating knowledge acquisition and performance systems. **Proceedings of the Australian Joint Artificial Intelligence Conference**. pp.317-336. Adelaide (November).
Shaw, M.L.G. & Gaines, B.R. (1987) KITTEN: Knowledge Initiation and Transfer Tools for Experts and Novices. **International Journal of Man-Machine Studies 27**(3), 251-280.
Shriver, B. & Wegner, P. (1987) **Research Directions in Object-Oriented Programming**. Massachusetts: MIT Press.
Trickey, H. (1988) C++ versus Lisp: a case study. **Usenix Procceding C++ Workshop**. pp.440-449. Berkeley: Usenix Association (November).
Voß, A., Karbach, W., Drouven, U., Lorek, D. & Schuckey, R. (1990) Operationalization of a synthetic problem. **ESPRIT Basic Research Project P3178 REFLECT Task I.2.1 Report** (July).
Watanabe, H. (1989) Heuristic graph displayer for G-BASE. **International Journal of Man-Machine Studies, 30**(3) 287-302 (March).
Woodward, J.B. (1990) Knowledge engineering at the front-end: defining the domain. **Knowledge Acquisition, 2**, 73-94 (March).