Int. J. Human–Computer Studies (1994) **41**, 59–107

Class library implementation of an open architecture knowledge support system

BRIAN R. GAINES

Knowledge Science Institute, University of Calgary, Calgary, Alberta, Canada T2N 1N4

Object-oriented class libraries offer the potential for individual researchers to manage the large bodies of code generated in the experimental development of complex interactive systems. This article analyses the structure of such a class library that supports the rapid prototyping of a wide range of systems including collaborative networking, shared documents, hypermedia, machine learning, knowledge acquisition and knowledge representation, and various combinations of these technologies. The overall systems architecture is presented in terms of a heterogeneous collection of systems providing a wide range of application functionalities. Examples are given of group writing, multimedia and knowledge-based systems which are based on combining these functionalities. The detailed design issues of the knowledge representation server component of the system are analysed in terms of requirements, the current state of the art, and the underlying theoretical principles that lead to an effective object-oriented implementation. It is shown that modeling the server through intensional algebraic semantics leads naturally to an open-architecture class library into which new data types may be plugged as required without change to the basic deductive engine. It is concluded that the development of a principled class library targeted on complex interactive applications does empower the individual researcher in the rapid prototyping of experimental systems. However, it is noted that much of the power of the approach stems from the cumulative evolution of the class library through successive applications, and hence the results may not generalize to team projects where greater rigidity is required in the class library in order to facilitate project management and inter-member coordination.

Introduction

One of the most difficult problems that faces researchers experimenting with complex interactive systems for real-world applications is the management of the large bodies of code generated in developing the systems. Personal computer users now commonly experience rich, multi-functional networked environments involving the integration of a wide range of applications. As the scope and functionality of such environments increase, it is becoming problematic to experiment with innovative applications since the development effort required can be very substantial while the utility of the research may be uncertain until empirical studies can be undertaken with a prototype system.

The object-oriented programming paradigm has been presented in the literature as providing capabilities for managing large bodies of code through structured modular decomposition that simplifies code generation and maximizes the possibilities for reuse (Booch, 1991). However, the effectiveness of the paradigm in

1071-5819/94/070059 + 49$08.00/0 © 1994 Academic Press Limited

providing support for large-scale software development through modularity and reuse has not yet been proven (Biggerstaff & Perlis, 1989*a*,*b*). It is to be expected that an object-oriented approach alone will not make a major difference to the problems of managing large-scale software development involving many people. The human factors of such projects often dominate the technological factors in determining what is critical to success, and it is now generally accepted that technology can only be used effectively when a high level of human process management has been achieved (Humphrey, 1989). For researchers, however, different questions apply, since the development resources available are generally small and the requirement is for powerful tools to support the individual researcher in rapidly prototyping complex systems that have to be sufficiently stable for purposes of experiment but do not, initially at least, require the full support of a commercial product.

This article describes experience in the application of the object-oriented paradigm to the development of knowledge support systems involving a wide range of complex, interactive technologies. The results suggest that a researcher, or small research team, can develop a class library that supports the rapid prototyping of a very wide range of systems including collaborative networking, shared documents, hypermedia, machine learning, knowledge acquisition and knowledge representation, and various combinations of these technologies. That is, the class library approach to development *does* empower the individual researcher and enable complex systems to be developed rapidly for purposes of experimental testing.

The following section gives an overview of the types of system that have been developed around the class library described, and the remainder of the paper focuses on one major application, that of the development of a principled, open architecture knowledge representation server (KRS) used to support many of the other applications. The server development is of particular interest because it demonstrates the links between the underlying theory of an application and its object-oriented implementation. It also exemplifies a technology that is of widespread application in many knowledge-based interactive systems.

Knowledge support systems

Expert systems, intelligent tutoring systems, knowledge acquisition systems, and a range of other knowledge-based systems, are examples of technologies supporting human knowledge processes. So are many other more routine technologies, such as electronic mail and desktop publishing. In system development we are moving towards the integration of a wide variety of such *knowledge support systems* developed by different research communities (Gaines, 1990*b*), for example:

- The *knowledge acquisition* community has generated requirements for systems to incorporate knowledge elicitation and analysis modules automating expertise transfer (Boose & Gaines, 1988; Gaines & Boose, 1988).
- The *machine learning* community has generated requirements for systems to incorporate inductive modeling modules automating empirical induction (Michalski, Carbonell & Mitchell, 1986).
- The *case-based reasoning* community has generated requirements for systems to

incorporate comparison modules automating analogical reasoning (Kolodner, 1988).

- The *natural language* and *machine translation* communities have generated requirements for systems to incorporate linguistic analysis and generation modules automating language understanding (Cullingford, 1986; Lehrberger & Bourbeau, 1988).
- The *knowledge representation, explanation-based learning* and *symbolic reasoning* communities have generated requirements for systems to incorporate deep reasoning modules automating reductionist derivation (Hobbs & Moore, 1985; Cercone & McCalla, 1987).
- The *intelligent tutoring* community has generated requirements for systems to incorporate user modeling modules automating individualized knowledge transfer (Wenger, 1987).
- The *electronic mail* and *teleconferencing* communities have generated requirements for systems to incorporate communication modules automating inter-person communication (Hiltz, 1984; Vervest, 1988).
- The *electronic publishing* community has generated requirements for systems to incorporate typographic and graphic modules automating knowledge presentation (Vliet, 1988).
- The *information retrieval* community has generated requirements for systems to incorporate knowledge indexing modules automating contextual retrieval (Oddy, Robertson, van Rijsbergen & Williams, 1981).
- The *visual programming* community has generated requirements for systems to incorporate interactive graphic modules automating visual interaction (Shu, 1988).
- The *hypermedia* community has generated requirements for systems to incorporate knowledge linkage modules automating associative thinking (Barrett, 1989).
- The *computer support of cooperative work* community has generated requirements for systems to incorporate knowledge-sharing modules automating group collaboration (Greif, 1988).

Each of these research communities has its own major specialties, conferences, literature and deliverables, and yet from a knowledge support systems perspective they are all closely inter-related in providing support of human knowledge processes. Moreover, from a user perspective the systems developed that provide services in each of the specialist sub-disciplines are complementary tools to which access should be provided through an integrated environment. One requirement of the research program of which the work described in this article is part has been to provide a software environment for research on knowledge support systems spanning the range of applications listed above, and their various combinations.

Figure 1 shows the architecture of the family of knowledge support systems for the Apple Macintosh supported by the class library discussed in this paper. It is shown as a "jigsaw puzzle" of loosely coupled components because that is the way the individual major sub-components have been designed, implemented and used. That is, in terms of overall functionality the system consists of a number of relatively independent applications that operate effectively as stand-alone programs, but can also be coupled together through inter-application protocols in various forms of integrated system. Underlying all the applications is a common class library with

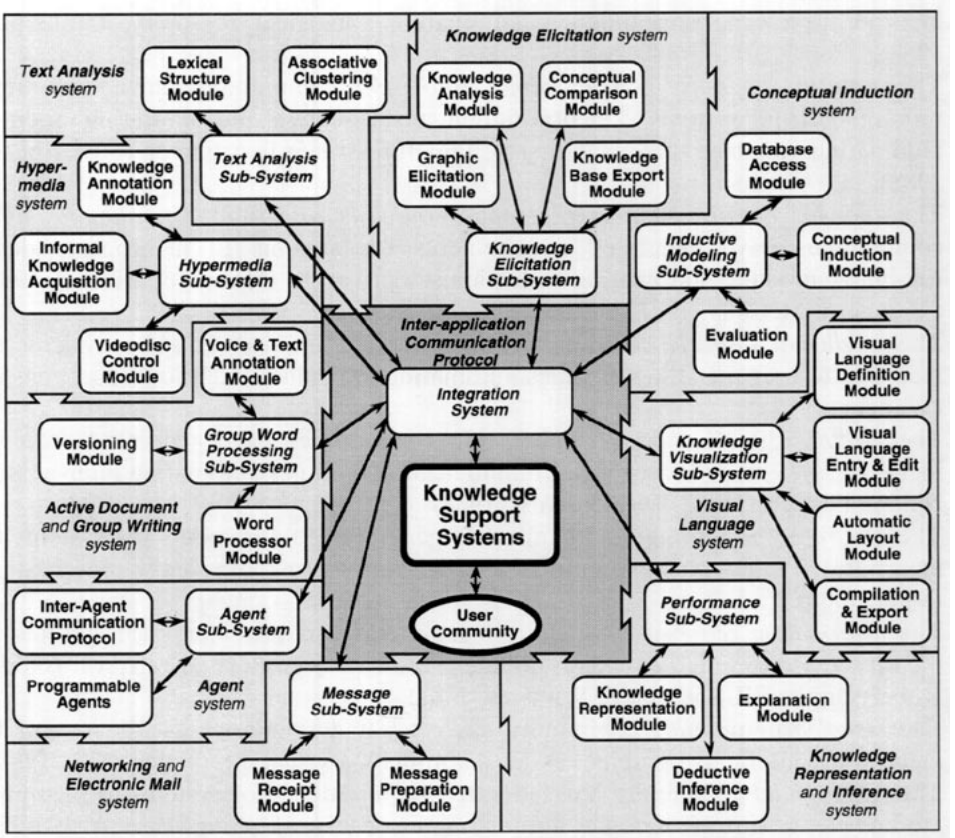

FIGURE 1. Architecture of a general knowledge support system.

modules that are reused in different ways in various applications. The class library provides all user interface functionality and hence supports the provision of a uniform style of access to all applications. It also provides filing, networking, inter-application communication protocols, and the wide range of data structures required by the different applications, and hence it exemplifies the possibility of factoring out common modules from many different applications and reusing them in apparently distinct implementations.

At the center of Figure 1 is the implementation of knowledge support systems serving specific user communities and each operating through a particular integration system that draws upon the services of the surrounding application sub-systems. A wide variety of knowledge support systems can be created with comparatively little effort by programming different integration systems. All of the application systems are themselves programmable through the Apple high-level object event protocol (Apple, 1993*a*), can be driven by any of the scripting languages in Apple's open scripting architecture such as AppleScript (Goodman, 1993), Frontier (Winer, 1992) and TCL (Ousterhout, 1994), and can themselves run scripts from within the application. The development of a specific knowledge support system generally involves writing scripts to provide the required functionality by drawing upon

existing application sub-systems. Highly innovative applications may require the existing applications to be extended or new ones to be written using the underlying C++ class library, and sometimes this may involve enhancement of the class library itself. However, as the family of application sub-systems has grown during the past four years, the prototyping of new experimental knowledge support systems has become increasingly a matter of writing a relatively small integration system as a script.

Reading around Figure 1 clockwise from the lower left corner, the major application systems are:

Networking and electronic mail system
The networking and mail sub-system supports both local and wide area networks through protocols such as AppleTalk and TCP/IP (Hunt, 1992). It also implements the Unix protocols (Krol, 1992) for telnet, ftp, smtp, gopher and world-wide web http, to allow integration with networked information retrieval and mail systems (Quarterman, 1990). An example application of this system is RepGrid-Net, a group knowledge elicitation and modeling tool that integrates knowledge elicitation methodologies with electronic mail (Shaw & Gaines, 1993).

Agent system
The agent sub-system supports independent light-weight processes across the network that can perform specific tasks such as transport of data for processing through specialist facilities at remote sites. It also implements the protocols for inter-agent communication such as Apple object events (Apple, 1993a) and KQML (Finin, Weber, Wiederhold, Genesereth, Fritzson, McKay, McGuire, Shapiro & Beck, 1992). An example application of this system is Mediator, a networked enterprise integration tool for coordinating the product life cycle in intelligent manufacturing systems (Gaines & Norrie, 1994).

Active document and group writing system
The active document and group writing sub-system supports the production and editing of documents with full typography and embedded pictorial material. It provides word processing and page layout features comparable with those of commercial applications. The documents are active in that they support embedded panes from other applications, and they are versioned at the paragraph and document levels to support collaborative writing. Example applications of this system are KWrite, a multi-media document preparation system (Gaines & Shaw, 1993*c*) which also supports embedded knowledge bases and hence can be used to run expert systems (Gaines & Shaw, 1992), and GroupWrite, a collaborative writing system (Gaines & Malcolm, 1993).

Hypermedia system
The hypermedia sub-system supports the acquisition, editing, linking and navigation of multimedia materials both within the computer and through control of external systems such as videodisks. An example application of this system is the study of the relative efficacies of books and hypermedia in the presentation of knowledge-structures for sports coaching material (Vickers & Gaines, 1988).

Text analysis system
The text analysis sub-system supports the analysis of text through a variety of tools for lexical analysis, assignment of parts of speech, assignment of affective loading (Whissel, 1989), and collocation analysis (Reed, 1984). An example application of this system is the Texan concept extraction and clustering tool used as a front-end to knowledge elicitation (Shaw & Gaines, 1987), and to model the conceptual structure of documents specifying the objectives of collaborative communities (Gaines & Shaw, 1994).

Knowledge elicitation system
The knowledge elicitation sub-system supports the development of conceptual structures through interaction with human experts using extended repertory grids. An example application of this system is the suite of knowledge elicitation tools comprising KSS0 (Gaines & Shaw, 1993*b*).

Conceptual induction system
The conceptual induction sub-system supports the development of conceptual structures through empirical induction from datasets of cases using the Induct algorithm (Gaines, 1989). It can derive a variety of structures, including rules with exceptions such as Compton's ripple-down rules (Compton, Edwards, Kang, Lazarus, Malor, Preston & Srinivasan, 1992). An example application of this system is the modeling of a large data set of some 47 000 medical cases comprising 10 years of data from a thyroid treatment clinic (Gaines & Compton, 1993).

Visual language system
The visual language sub-system supports the specification and application of arbitrary visual languages comprising graphs of nodes and links with a wide variety of node and link types available. It enables a system developer or an end-user to create customized tools for concept mapping, semantic networks, bond graphs, Petrinets, and so on, without having to develop code. Example applications of this system are the KDraw semantic network tool (Gaines, 1991*b*) and the KMap concept mapping tool (Gaines & Shaw, 1993*d*).

Knowledge representation and inference system
The knowledge representation and inference sub-system supports knowledge bases and object-oriented and deductive databases. An example application of this system is the solution of the Sisyphus room allocation problem (Gaines, 1994).

Examples of implemented systems

The jigsaw puzzle diagram of Figure 1 indicates how complex system architectures may be developed by integration of a heterogeneous collection of specific application sub-systems. However, it does not show the functionality of a particular application or how much of this is drawn from the standard modules in the object-oriented class library. This section describes three example applications to illustrate the use of the class library.

GROUP WRITING EXAMPLE APPLICATION

The active document and group writing system at the center left of Figure 1 provides an illustration of the issues involved in developing a particular module, particularly since it involves encapsulating major assembly language modules that are not part of the object-oriented architecture. The design requirements for this module are to provide a word processing and page makeup system that has similar functionality to commercial word processors, a user interface that is familiar to users of such systems, and enhanced capabilities for hypertext, hypermedia, versioning and active components that are accessible through natural extensions to the interface.

GroupWrite (Malcolm, 1991; Gaines & Malcolm, 1993) is a word processor built using this module which was designed to support a geographically dispersed community of collaborating authors. The users are assumed to be jointly producing documents through simultaneous editing but exchanging material by email or through disks without the interlocks that require continuous network communication. GroupWrite versions documents at both the document and at the paragraph level such that one document may be opened in the context of other versions of it, and the editing and alternative versions are naturally visible and simply accessible to the user. It also provides facilities for textual and sound annotations attached to paragraphs.

Figure 2 shows a screen dump of an author editing a document in GroupWrite and requesting that the result of merging two different versions of the same paragraph be shown. Note that the word processor interface itself is similar to those of commonly used systems, providing control over typography, margins, indentation, tabs, justification, and so on. Authors can use the word processing facilities in a standard way with no need to be aware that there are additional features. Documents can be opened in the normal way through an "Open" dialog in the "File" menu, or through a graphic document version browser as shown in Figure 2 that utilizes the visual language module at the center right of Figure 1.

The document shown in Figure 2 has been opened by clicking on a previous version shown in reverse video in Figure 3, and then double clicking on a later version to open it in the context of the selected version or versions. GroupWrite compares the version information stored with the documents and indicates with a side marker any paragraph in the open document that has one or more alternative versions in the contextual documents. The appearance of the line varies depending on what types of annotation are present, and whether alternative paragraphs are present. The mouse pointer changes shape when moved over the line, to indicate that a pop-up menu is available. This menu may also be accessed by holding down the option key while clicking the mouse in the marked paragraph.

The pop-up menu provides a number of operations that may be performed on annotations and alternative paragraphs. They may be displayed in a separate window by the *Show* operation; inserted into the document text by the *Place Before*, *Place After*, or the *Replace* operations; and removed by the *Remove* operation. Sound annotations may be heard by selecting the *Play* operation. Annotations and alternative paragraphs are selected from the sub-menu, with annotations appearing below the dotted line.

Figure 4 shows the implementation architecture of GroupWrite as a three-layer

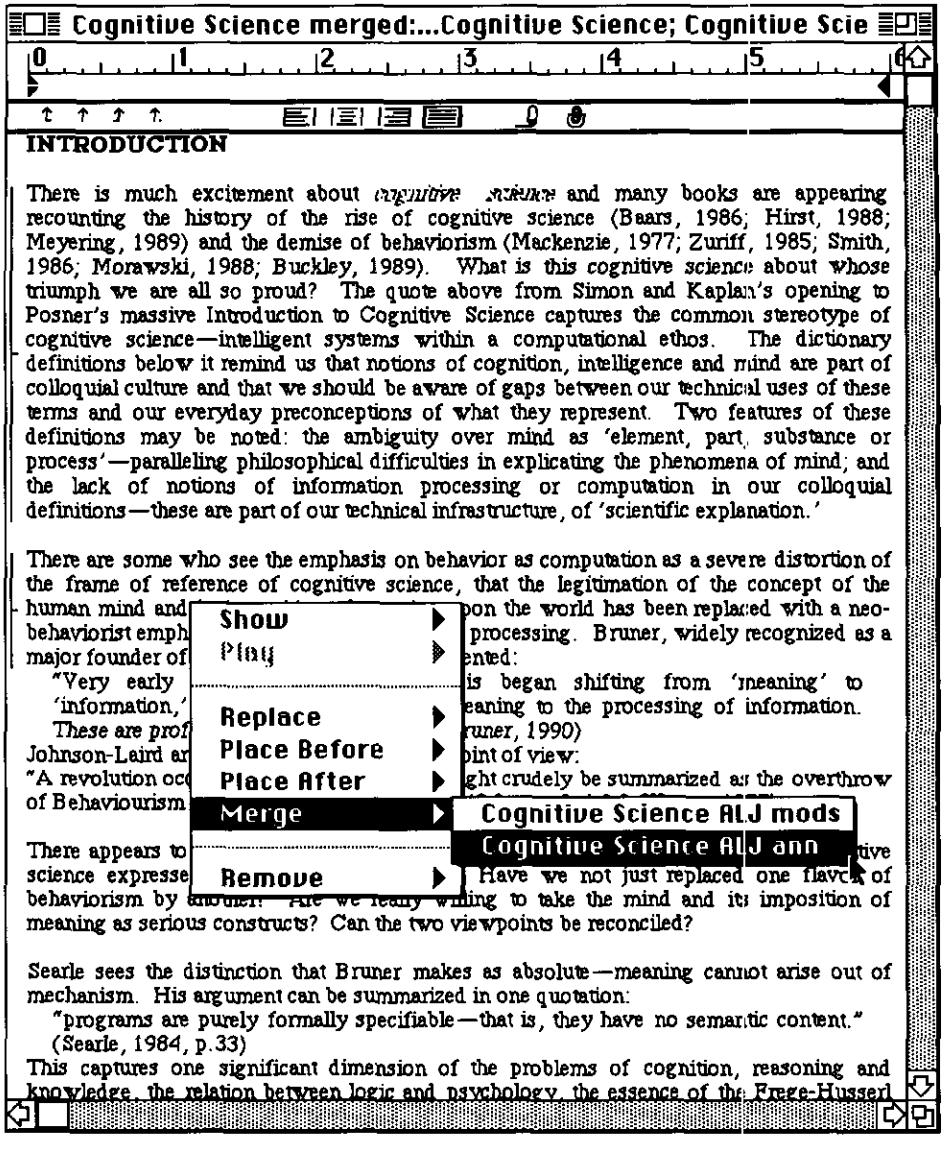

FIGURE 2. GroupWrite paragraph version markers and pop-up menu access.

structure with the top layer comprising modules specifically written to support GroupWrite features, the center layer comprising modules common to most applications, and the lowest level the assembly language modules of the word processing engine.

The top-level functionality is provided by GroupWrite-specific sub-classes of the class library, which may be grouped into four major components:

- Document management supporting the users' concept of a single document file as the basic unit which is edited and communicated.
- Editor support through a simple and natural user interface that, as well as

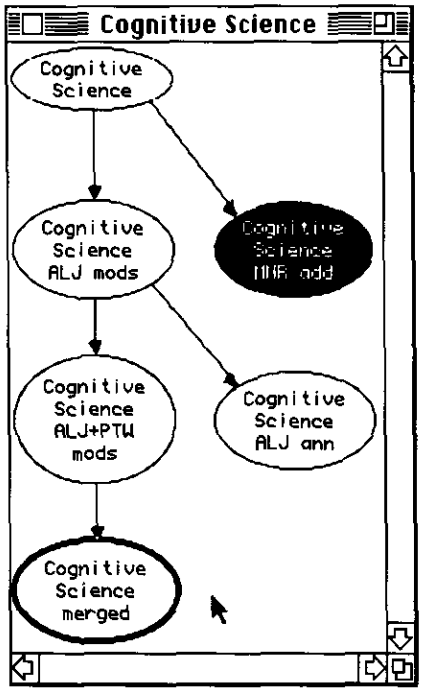

FIGURE 3. GroupWrite document version browser.

maintaining the content of the document and its annotation, also keeps track of what paragraphs have been changed for versioning purposes.

- Filing facilities that, as well as storing the document and annotation, also store tables of version information as separate resources in the same file as the document.
- Interchange facilities that allow the import and export of documents in RTF format allowing users to move files between GroupWrite and commercial word processors.

The middle-level functionality is provided by the standard class library, and in particular four classes of functionality significant to the GroupWrite implementation:

- Multi-media document support providing an open architecture word processor as a text editing module that supports full typography, picture placement, pagination and paragraph version tracking.
- Graphic user interface support for drawing and keyboard and mouse interaction with text editing windows, dialogs, and interactive graphics.
- Complex data structures support for a wide range of variable-size abstract data types including arrays of variable-size structures.
- General operating system support making available the range of managers available in the Macintosh Toolbox encapsulated as classes, including filing and inter-application communication.

The bottom-level functionality is the provision of the word processing facilities, which is implemented at the lowest level in terms of calls to a commercial software

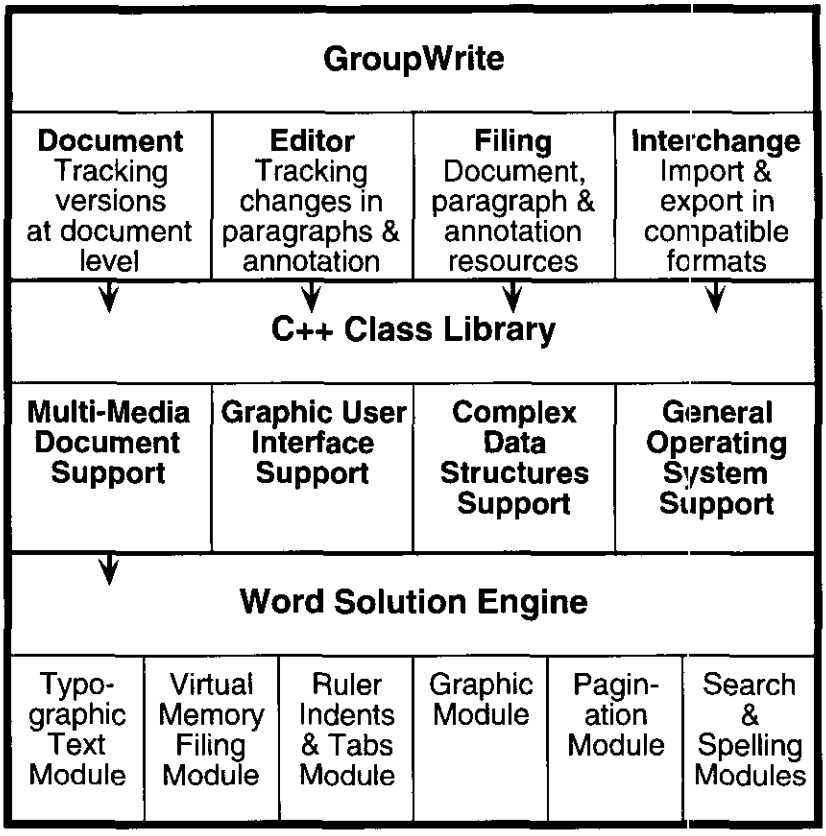

FIGURE 4. GroupWrite implementation architecture.

product, Datapak's Word Solution Engine (Crandall, 1990). This is written in M68000 assembly language for speed and provides a very comprehensive set of word processing functions through a basic module and a series of add-on modules providing additional functionality. The main modules are:

- Typographic text module supporting display and editing of text with varying fonts and sizes.
- Virtual memory filing module supporting the buffering of large documents to be editable within a specified memory allocation.
- Ruler indents and tabs module supporting variable paragraph width, line spacing, tab placement and boxing.
- Graphic module supporting the placement and sizing of graphics placed in the text.
- Pagination module supporting the display and output of text pages.
- Search and spelling modules supporting the normal word processor capabilities of searching for and replacing phrases, and checking the spelling of words.

The assembly language modules are encapsulated by C++ wrapper code so that users of the class library see a normal class structure with data structures and

methods that can be sub-classed, and are unaware of the underlying low-level implementation.

MULTIMEDIA, VISUAL LANGUAGE AND DOCUMENTATION EXAMPLE APPLICATION

The integration of multiple heterogeneous modules to provide a single application can be illustrated in the context of document systems by a multimedia application that combines the visual language, multimedia and active document systems. The class library provides an abstract "pane" class which generally supports the visual presentation of material within a window. The word processing class allows such panes to be embedded within a document in such a way that each pane component is relatively independent of the text functionality. The text is laid out to run around or hop over pane areas but is not otherwise affected by them.

The embedded panes may be used to display graphic material embedded in the document including making graphics available to other applications as if each were a drawing pane in an arbitrary window. This enables them to be used to support visual activity ranging from bit map and line graphics to QuickTime (Drucker & Murie, 1992) or laserdisk videos, through simulation and animation, to graphic editors for visual languages representing programs, concept maps, and formal knowledge structures. Mouse down-clicks within a pane are reported to the associated application rather than to the document software, and hence user interaction can be supported in a completely different environment. Such functionality is now becoming commonly available through the OLE 2 (Microsoft, 1994) and OpenDoc (Apple, 1993*b*) operating system extensions and allows many applications to be combined in a compound document.

Figure 5 shows a concept map of a knowledge structure developed for coaching ice hockey (Vickers, 1990) embedded in a document together with a QuickTime video showing the relevant activities. The material relates to some earlier studies providing a detailed comparison of laserdisks and books as media through which to present the coaching material (Vickers & Gaines, 1988). When the multimedia document technology became available, some of the video material was digitized and incorporated together with text and active knowledge structures in documents that emulate sections of the existing books and laserdisk material. At the top of Figure 5 is the relevant part of the overall knowledge structure as an active concept map. As the user mouses over a node in the map a pop-up menu symbol appears. Clicking on this sends a message to a script associated with the document that determines the menu items, in this case the capability to play a video clip. The selection of an item sends a message to the script which then plays the appropriate section of the QuickTime movie below the knowledge structure.

Producing the document shown in Figure 5 is a simple matter of selecting the "Place" dialog in the "File" menu and selecting the files for the concept map and the video respectively. The concept map is automatically active and can be edited in place, including entering the links to the appropriate sections of the movie. The underlying script for the multimedia documents provides a generic linkage facility and does not have to be edited by the document producers. However, it is accessible to them if they wish to enhance the facilities in some way.

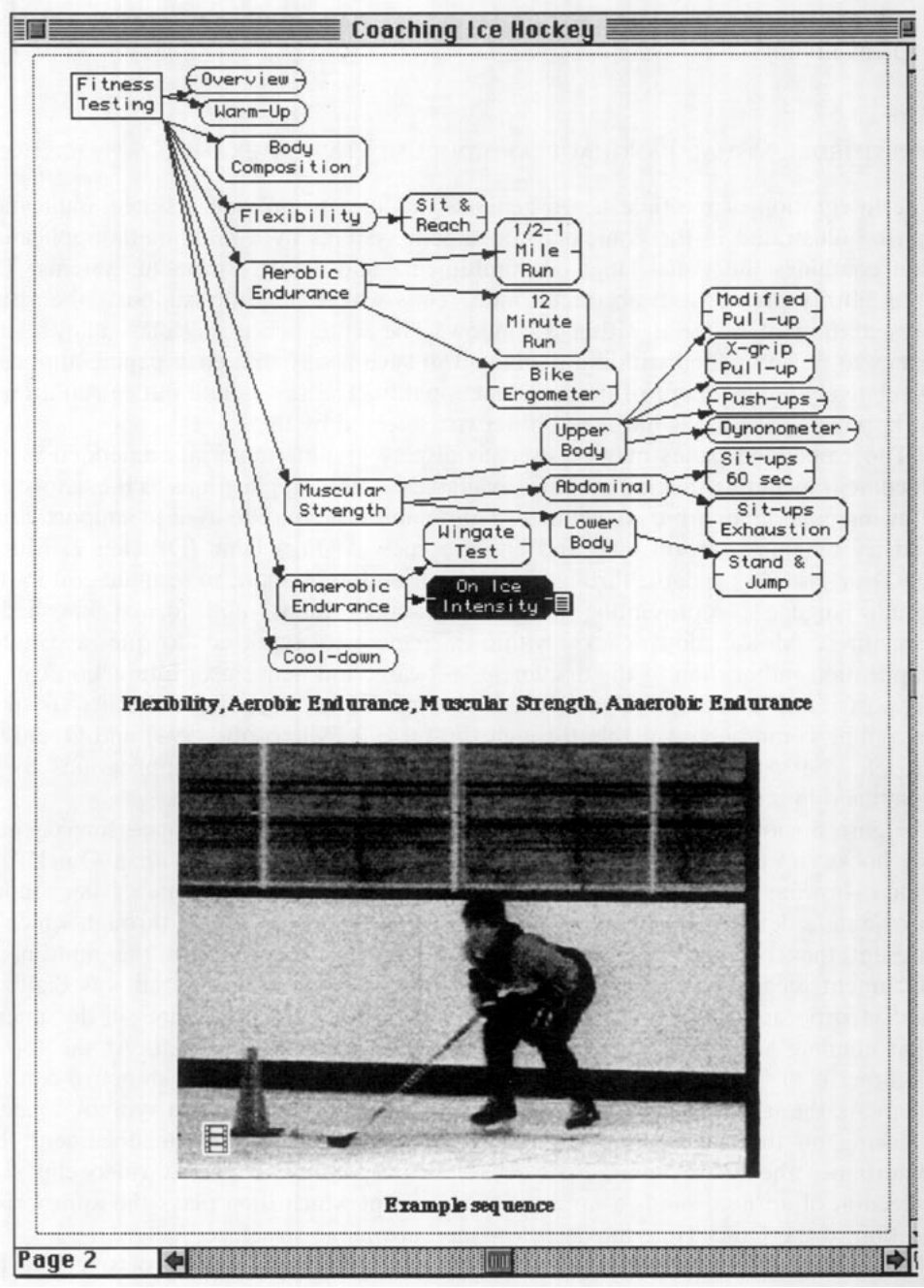

FIGURE 5. Concept map of coaching knowledge structure linked to video exemplars.

SEMANTIC NETWORK, EXPERT SYSTEM AND DOCUMENTATION EXAMPLE APPLICATION

The embedded active concept maps of the multimedia example extend naturally to provide a visual language for KL-ONE knowledge structures (Gaines, 1991*b*), providing formal knowledge representation that can be compiled and run within the knowledge representation and inference system at the bottom right of Figure 1. This provides an illustration of the integration of the multimedia and artificial intelligence components of Figure 1, and also provides a bridge to the detailed analysis of the knowledge representation part of the class library in the remainder of this paper.

Figure 6 shows a screen dump from KWrite editing a paper that was published in the proceedings of the British Computer Society Expert Systems Conference in December 1992 (Gaines & Shaw, 1992). The paper puts into active document form a knowledge-based system developed as a solution to a "challenge" problem circulated by Project Sisyphus, an initiative of the European Knowledge Acquisition Workshop. The problem is one of room allocation derived from an ESPRIT project (Voß, Karbach, Drouven, Lorek & Schuckey, 1990), and a major part of the EKAW'91 program was devoted to reports on the solution of this problem using different approaches and techniques (Linster, 1991).

The abstract of the BCS paper describes the form of publication:

> This paper is written in a document production tool that appears to a user as a word processor but also acts as an expert system shell with frame and rule representations supporting deductive inference. The electronic version of the document is active, providing typographic text and page layout facilities, versioning, hypermedia sound and movies, hypertext links, and knowledge structures represented in a visual language. It can be read as a hypermedia document and also interrogated as a knowledge-based system for problem-solving. The paper version of the document, which you are now reading, is produced by printing the electronic version. It loses its active functionality but continues to act as a record of the knowledge in the document. The overall technology has been developed as an alternative approach to the dissemination of knowledge bases. It also provides a different interface to knowledge-based systems that emulates document interfaces with which many users are already familiar. (Gaines & Shaw, 1992: p. 331)

The electronic version of the paper was made available through anonymous ftp and as a CD-ROM. It demonstrated parallel publication of the paper version as a camera-ready copy, book copy and the electronic version, identical in appearance, as a full working demonstration of the problem solution. What is particularly significant is that there were no hidden data structures. The semantic networks in the paper were the complete knowledge structures and operated directly in the inference engine to solve the problem. Thus, the document was also the knowledge base and editing the knowledge structures within the document could change the ontologies, rules or data, and hence the outcome of inference.

The editing facilities for embedded components are provided through a class library facility that allows dialogs that would normally appear at the top of a window associated with editing a particular component to be accessed as a floating dialog when that component is embedded in a document. Thus, the page shown in Figure 6 appears as a normal word processing document page until the user double clicks in one of the semantic network "pictures". Then the floating dialog appears, enabling the network to be edited in place exactly as if it were in the visual language tool that originally produced it.

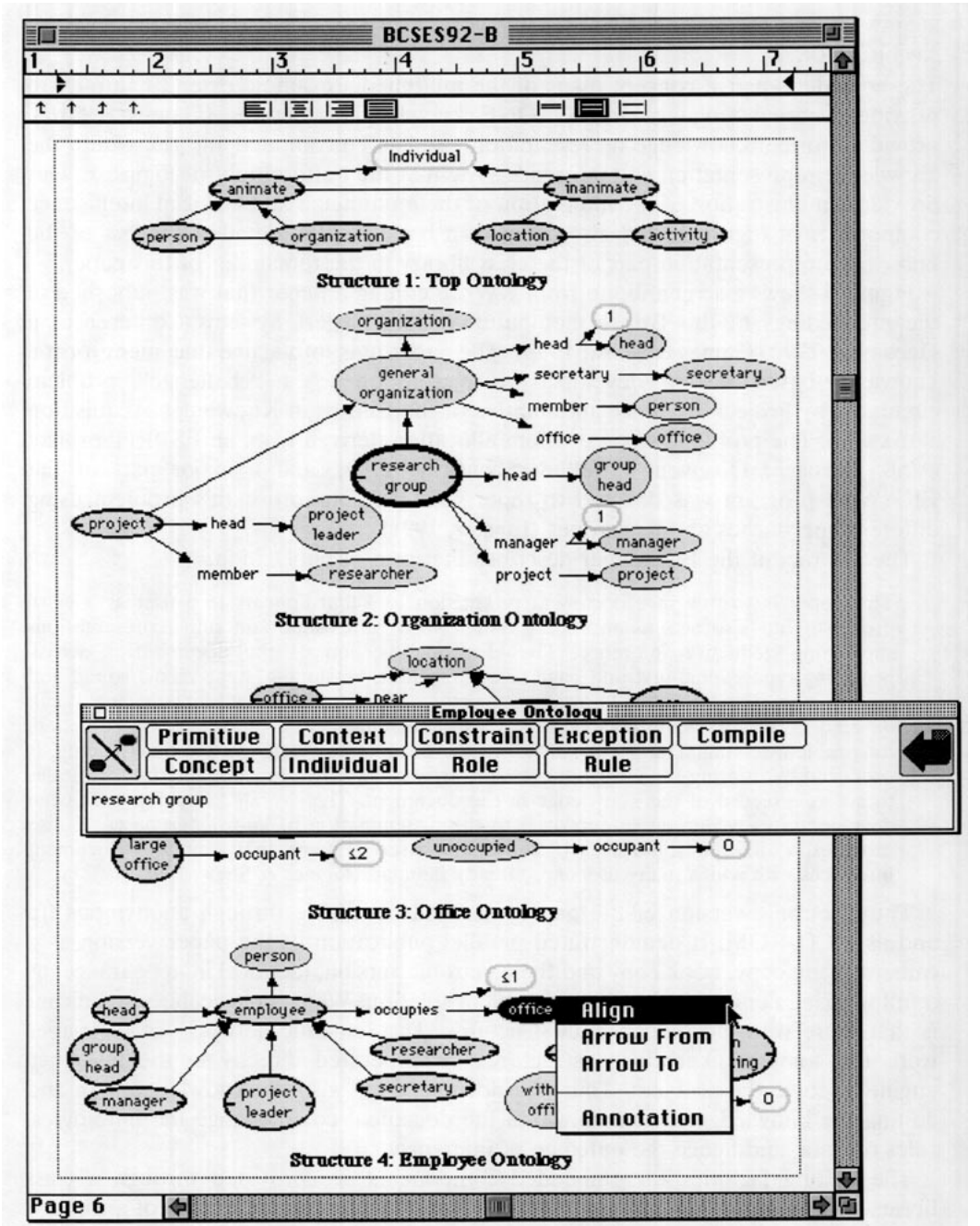

FIGURE 6. Interaction with knowledge structures in an active document.

The multimedia and pop-up menu linkage facilities remain available, and, for example, the user is shown at the bottom right of Figure 6 as having accessed the pop-up menu associated with a particular node. One of the facilities this makes available is access to annotation for that node, and, since the script has full operating system access, this may be within the same document, in another application on the machine such as HyperCard (Goodman, 1990), or in another application across the network, possibly in another country.

Knowledge representation

The article so far has presented the application level and its relation to the underlying class library. The remaining part will focus on the detailed design of a significant part of the class library, that associated with knowledge representation. This part was selected for detailed discussion for a number of reasons. First, because class libraries for graphic user interfaces are already discussed in depth in the literature (Pinson & Wiener, 1990; Wisskirchen, 1990; Bass & Dewan, 1993). Second, because knowledge representation is a basic service that may be factored out of virtually all knowledge support systems, and plays a major role in both artificial intelligence and human–computer interaction. Third, because the design of the knowledge representation of the class library shows the deep interplay between the theoretical issues of formal representation and the practical issues of implementation. It seems likely that such relations between theory and practice will become increasingly significant as formal methods are applied to the routine development of interactive systems (Thimbleby, 1990).

The initial requirement for a knowledge representation module in the knowledge support system class library originated in 1988 as part of studies of repertory-grid and induction-based knowledge acquisition tools. The repertory grid is a constructivist conceptual modeling methodology that was proposed by Kelly (1955) in the mid 1950s as a clinical technique, became widely applied in psychology, education and management (Shaw, 1981), was implemented through interactive computer programs in the mid 1970s (Shaw, 1979, 1980), and was proposed as a technique for knowledge elicitation for expert systems in 1980 (Gaines & Shaw, 1980). In the early 1980s the technique became used for knowledge acquisition (Shaw & Gaines, 1983; Boose, 1984), but the simple underlying entity–attribute representation of the original grid was also substantially enhanced (Boose & Bradshaw, 1987; Boose, Bradshaw, Kitto & Shema, 1989) in ways that gave new insights into both the underlying personal construct psychology and the general principles of knowledge representation (Gaines & Shaw, 1993*a*). Such studies led to a requirement for an open-architecture, principled knowledge representation server that could provide knowledge representation services to a wide range of knowledge acquisition tools.

The objective of knowledge acquisition research in the 1980s has been primarily to develop knowledge structures that could be transferred to expert system shells such as NEXPERT (Gaines, Rappaport & Shaw, 1992) and Babylon (Gaines & Linster, 1990), and the obvious basis for a knowledge representation server was some form of expert system shell. However, commercially available shells have been developed to satisfy the wide ranging requirements of a diverse community of customers, and did not offer principled architectures which could be understood in terms of

underlying logical processes. The most appropriate starting point appeared to be the work on KL-ONE-like systems, variously called *term subsumption* or *terminological* systems, or *description logics*, since they focus on the formal definition of conceptual structures and on the deduction of the subsumption relations between them (Nebel & Smolka, 1990; Schmolze & Woods, 1992).

The KL-ONE family of knowledge representation has its origins in early representation schema in the mid-1960s in terms of semantic networks (Quillian, 1968), which were attractive in terms of their visual representation of knowledge structures but had problems of imprecise semantics which were analysed by Woods (1975) and Brachman (1977) in the mid-1970s. In the 1980s the formal foundations of such systems were developed in terms of intensional logics (Maida & Shapiro, 1982) and complexity theory (Brachman & Levesque, 1984), and during the 1980s increasingly principled system designs were developed, such as KL-ONE (Brachman & Schmolze, 1985), KRYPTON (Brachman, Gilbert & Levesque, 1985), BACK (Nebel, 1990), LOOM (MacGregor, 1991), CLASSIC (Borgida, Brachman, McGuiness & Resnick, 1989) and KRIS (Baader & Hollunder, 1991). Deep theoretical foundations have been developed for such technologies in recent years (Aït-Kaci, 1986; Nebel, 1990). The availability of these foundations makes it possible to develop general-purpose knowledge representation services that are well-principled, space and time efficient, and embeddable as sub-systems in a wide variety of applications.

Since complexity analyses show that inference in knowledge representation systems with normally expected representational capabilities is intractable (Brachman & Levesque, 1984; Nebel, 1988; Schmidt-Schauß, 1989), it has been suggested that the representational and deductive capabilities of knowledge representation services should be limited for the sake of tractability (Levesque & Brachman, 1987). The reasons for and against this have been surveyed by Doyle and Patil (1991), who conclude that, while there are sound arguments for such limitations, the capabilities of the resultant systems will often fail to satisfy reasonable application requirements. One impact of this is that system designers may add functionality that provides the missing capabilities in ways that are less principled than those of a general server. A second is that problems may have to be represented in unnatural ways that are conducive to poor performance. A third is that the effect of the limitations on deductive capabilities may not be apparent to users, leading to errors.

These arguments were very relevant in the context of knowledge acquisition and knowledge support systems since it was precisely the need to extend existing knowledge representation schema that motivated the design of a knowledge representation server, and yet it was important that the system designed be fast and efficient for interactive application on personal computers. These considerations motivated an *open-architecture* server design in which the kernel system would provide basic capabilities known to be both time and space efficient, but to which functionality might be added in a principled fashion. This in turn motivated implementation as an object-oriented class library with well defined interfaces to new classes supporting additional data types. In particular, the knowledge acquisition experience led to requirements for the data types common in databases, such as integers, floating point numbers, dates and strings, which were not treated in a principled fashion in the theory of KL-ONE-like knowledge representation systems.

It became apparent that a constraint-theoretical model provided theoretical foundations for both the original KL-ONE features and the extensions to new data types, and this was taken as the basis for a principled design (Gaines, 1993).

The remainder of this paper describes the design and implementation of the knowledge representation and inference system shown at the bottom right of Figure 1 as an open architecture class library in C++.

Knowledge representation server (KRS) design requirements

The basic requirement was for a knowledge representation and inference system that could represent abstract concepts and their subsumption relationships, concrete cases with attributes, values and relationships, and inference rules for deriving further information about cases from their asserted properties. This broad requirement is currently satisfied by three convergent forms of technology: object-oriented databases (Gupta, 1991); deductive databases (Minker, 1988); and a variety of knowledge representation schema based on semantic networks (Sowa, 1991), such as KL-ONE (Brachman & Schmolze, 1985), conceptual graphs (Sowa, 1984) and SNePS (Kumar, 1990). CLASSIC (Brachman, McGuiness, Patel-Schneider, Resnick & Borgida, 1991) was selected as a starting point for design because it was well-defined in syntax and semantics, had major real-world applications (Devanbu, Selfridge, Ballard & Brachman, 1989; Wright, Weixelbaum, Vesonder, Brown, Palmer, Berman & Moore, 1993), defined a well-integrated set of kernel features that were an adequate foundation for knowledge acquisition, and could be regarded as an extended object-oriented database (Borgida *et al.*, 1989) or a restricted knowledge representation system (Patel-Schneider, McGuiness, Brachman, Resnick & Borgida, 1991). The initial implementation of KRS with the full set of CLASSIC 1 capabilities using a published paper (Borgida *et al.*, 1989) as a design specification took five person-weeks in C++ (Gaines, 1990*a*).

CLASSIC 1 (Resnick, Borgida, Brachman, McGuiness & Patel-Schneider, 1990) has a compositional semantics in which complex knowledge structures are composed from eight simple semantic structures:

- *Concepts* are abstract objects, each defined completely by its compositional semantics.
- *Primitive concepts* are concepts in which the overt definition is incomplete and an additional non-overt component is assumed. A relation between the non-overt components may be specified to be such that two primitive concepts are *disjoint*, i.e. such that their composition is incoherent.
- *Roles* capture the semantics of attributes and relations.
- *Individuals* are concrete objects, each defined by their specified identifier, and each having a variable state specified by asserting that the individual is an instance of a concept.
- *Cardinality concepts* are specified in terms of lower or upper bounds on the size of sets.
- *Explicit extensional concepts* are specified in terms of sets of individuals as lower or upper bounds upon other sets.

- *Implicit extensional concepts* are specified in terms of equality relations between sets of individuals (*coreference* in CLASSIC).
- *Rules* are specified as a pair of concepts such that when the premise concept applies to the state of an individual the conclusion concept may also be asserted to apply.

KL-ONE implementations group the conceptual definitions into a terminology module, the *T-box*, the assertions about individual states into an assertional module, the *A-box,* and the rules into a rule module, the *R-box*. If the implementation reasons over possible different states it is necessary to introduce a fourth module, tracking possible worlds (Filman, 1988), and this may be termed a *W-box*.

CLASSIC 1 made provision for additional data types such as numbers by allowing them to be specified as "host individuals" and providing a *test* concept that allows an arbitrary test to be applied to an entity, such as a number, that is not representable in terms of the basic semantic structures defined above. Knowledge acquisition applications typically require representation of integers, reals, dates, and so on, and these were so fundamental to the representation that it was undesirable to capture them through arbitrary tests that fell outside the well-defined semantics of the other representational structures. Hence, the possibility of extending CLASSIC's representation to encompass additional data types was investigated, and the way in which this was done and its implementation in the class library are described in the next section. The theoretical foundations for adding concrete domains to knowledge representation languages have been developed by Baader and Hanschke (1991), and CLASSIC 2 (Resnick, Borgida, Brachman, McGuiness, Patel-Schneider & Zalondek, 1993) supports concepts expressing interval bounds on numbers.

Other extensions were made to CLASSIC 1 that were found necessary in knowledge acquisition applications. First, the inverse relation between roles was made definable: that, for example, "child" is inverse to "parent". Second, the specification of sets through negation was added: that, for example, "Not Fred or Mary" defines a set. This addition is important to support non-closed-world semantics where the complements of well-defined sets cannot be enumerated, so that one cannot specify "Not Fred or Mary" positively by enumerating all the individuals who are not Fred and not Mary. Third, recognition of individuals as instances of concepts was extended to take into account the properties of other individuals filling roles in those concepts. Fourth, provision was made for conceptual constraints to be applied to concrete data types such as integers, in order to specify units, such as amount in $, or weight in kg. Fifth, the specification of implicit extensional concepts by equality (coreference) was extended to specification by inclusion: for example, that the friends of one individual include the friends of another. However, it should be noted that reasoning with implicit extensional constraints in the current implementation of the T-box in KRS is incomplete. It should also be noted that inverse roles, and other features such as role hierarchies, have been added to CLASSIC 2 (Resnick *et al.*, 1993), and Borgida and Patel-Shneider (1994) have published an algorithm for subsumption computation in CLASSIC 2 that is proved to be complete under certain restrictions.

Inference in CLASSIC consists of propagating conceptual constraints by composing those that form the state of an individual through assertions made about the

individual and the rules that apply to it. There are two major relations between concepts that may be computed:

- Subsumption, that one concept subsumes another if their composition is the same as the second concept.
- Incoherence, that two concepts are mutually incoherent if their composition is logically contradictory.

The definition of subsumption may seem converse to the colloquial usage of the term. This is because the technical usage is in terms of extensional semantics. If one concept subsumes another then the *set of the individuals* whose states are subsumed by the first concept *includes* those whose states are subsumed by the second. Whereas, from an intensional perspective, the *properties* of the first concept are *included in* those of the second.

The computation and caching of the subsumption relation between concepts is significant to the speed of rule-based inference. It speeds up the detection of individuals whose state is such that a rule applies. It is also important in knowledge acquisition systems such as KREME (Abrett & Burstein, 1988) as a basis for re-representing a conceptual structure to display conceptual relations that may not have been intended by the expert and may indicate errors.

The computation and caching of the incoherence relation between concepts is significant to the speed of constraint-based inference. It speeds up the detection of individuals whose state has become incoherent through assertions that are thereby deemed unacceptable and have to be retracted. This is important in applications of CLASSIC-like systems to managing the software engineering process (Devanbu *et al.*, 1989), or to the management of the relation between activities and objectives in large-scale projects (Gaines & Shaw, 1994). It is also important in knowledge acquisition systems such as MOBAL (Morik, 1989) as a basis for detecting inconsistencies in a conceptual structure or its usage which may indicate errors.

KRS implementation architecture and class library

The initial class library for an implementation of KRS on the Apple Macintosh was developed as an extension of the THINK class library, originally using object-oriented extensions to C which was later extended by Symantec to provide a full C++ level 3 compiler. The THINK library is designed to support graphic user interfaces and operating system access, and provides only the few data types necessary to do this. The major part of the KRS development consisted of developing appropriate data structures and was thus relatively independent of the existing library.

Figure 7 shows the main data structures in KRS:

- *Concept records* hold the constraints defining a concept: its primitives and those disjoint from them; its extensions as the sub-sets of individuals that must be, and may be, included in any sub-set recognized; its cardinality and inclusion constraints on any sub-set recognized; its rule concept, if any, which will be asserted of any individual recognized; and, for each role constrained in the concept, the concept constraining it.

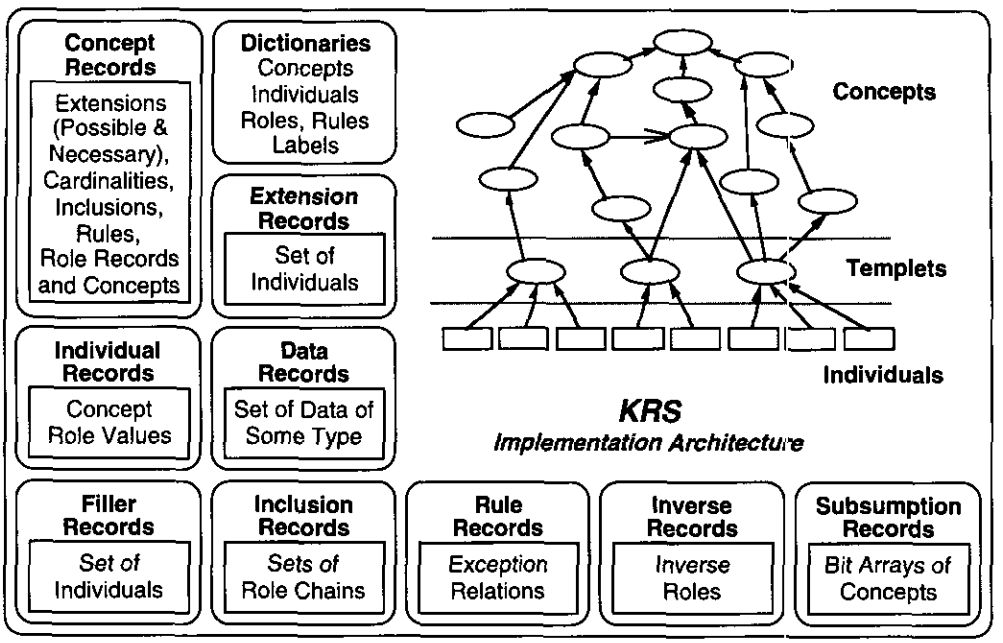

FIGURE 7. Data structures in the knowledge representation server.

- *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 set of role fillers.
- *Filler records* hold sets of individuals that fill roles.
- *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 sub-sets of individuals, typically those that may occur in a role filling set and those that must occur in it.
- *Data records* hold sub-sets of external individuals that are explicitly recognized by the server, notably integers, reals, dates and strings.
- *Inclusion records* hold relations between role chains expressing inclusion and coreference constraints.
- *Rule records* keep track of exception relationships between rules (Gaines, 1991*a*).
- *Inverse records* keep track of inverse relations between roles.
- *Subsumption records* hold computed subsumption and incoherence relations on a three-valued basis, that one concept subsumes another, does not, or cannot.

The implementation of the A-box 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. For example, it makes it clear when an assertion about

an individual has been both asserted directly and derives from a rule so that the retraction of the direct assertion will not change the state of the individual.

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.

Figures 8(*a*) and 8(*b*) show the sub-class structure in the THINK class library together with the additional classes written to implement the knowledge support systems of Figure 1. The classes discussed are tinted so that they may be more easily distinguished. The classes particular to KRS are primarily those in Figure 8(*a*) that sub-class the **Data** class on the left.

KBParse at the lower center right is a KRS knowledge base with the sub-class chain:

Object—the THINK class library top-level class of object 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 concept definitions
KBIO—adding input/output functionality
KBParse—adding language parsing functionality

KBParse uses component objects that are sub-classes of **Items** such as:

KBIndividuals—KRS individual data
KBInfer—dependencies between KRS objects (for rapid retraction)
Sets—set data records as shown in Figure 7
Alias—dictionaries of names as shown in Figure 7

It also uses component objects that are sub-classes of **Data** such as:

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

A significant user interaction class in the top half of the diagram is **KBEditDoc** at the top right of Figure 8(*b*). It implements KRS's knowledge visualization sub-system shown at the right of Figure 1. **KBEditDoc** is at the end of a long sub-class chain:

Object—the THINK C top-level class of object definition
Collaborator—a class for objects that have mutual dependencies
Bureaucrat—a class for objects that manage other objects
DirectorOwner—a class for objects that manage directors
Director—a class for objects that manage a window

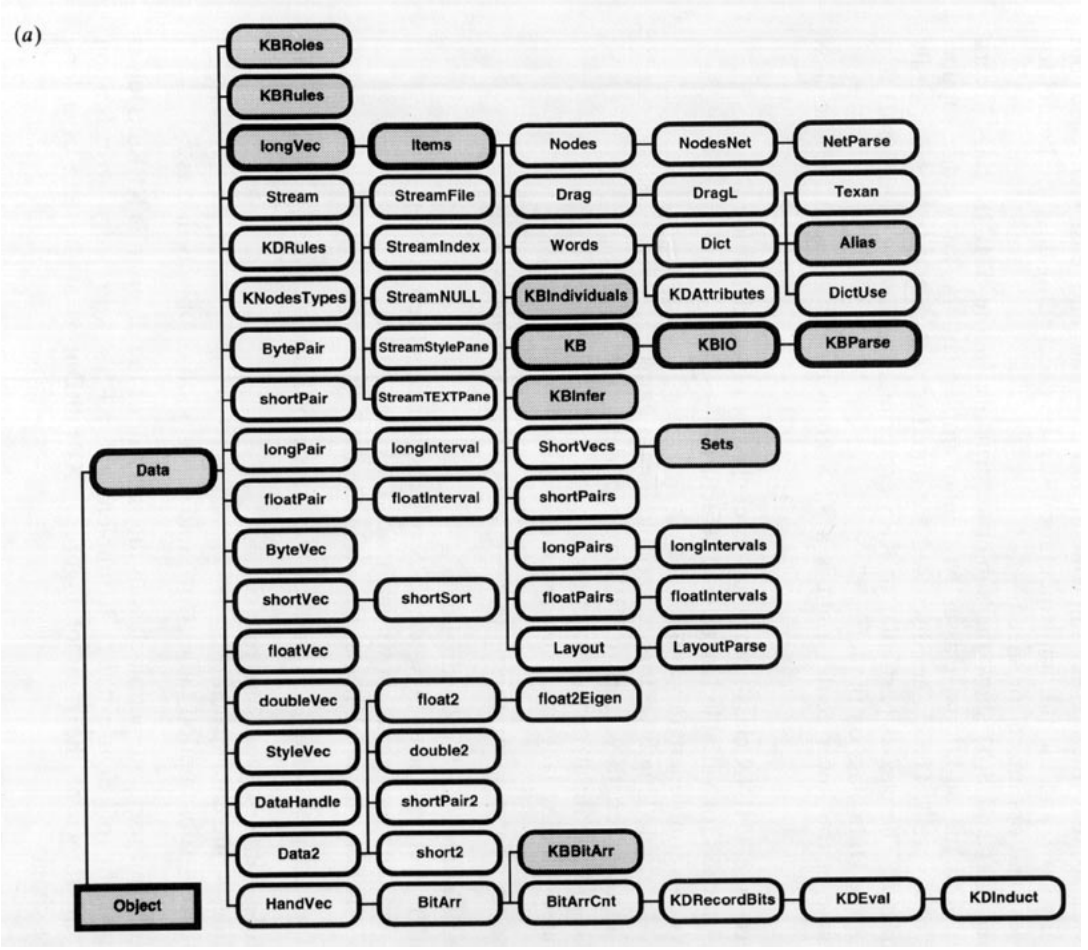

FIGURE 8. (*a*) Knowledge support system class library—data structures; (*b*) knowledge support system class library—user and operating system interface.

(b)

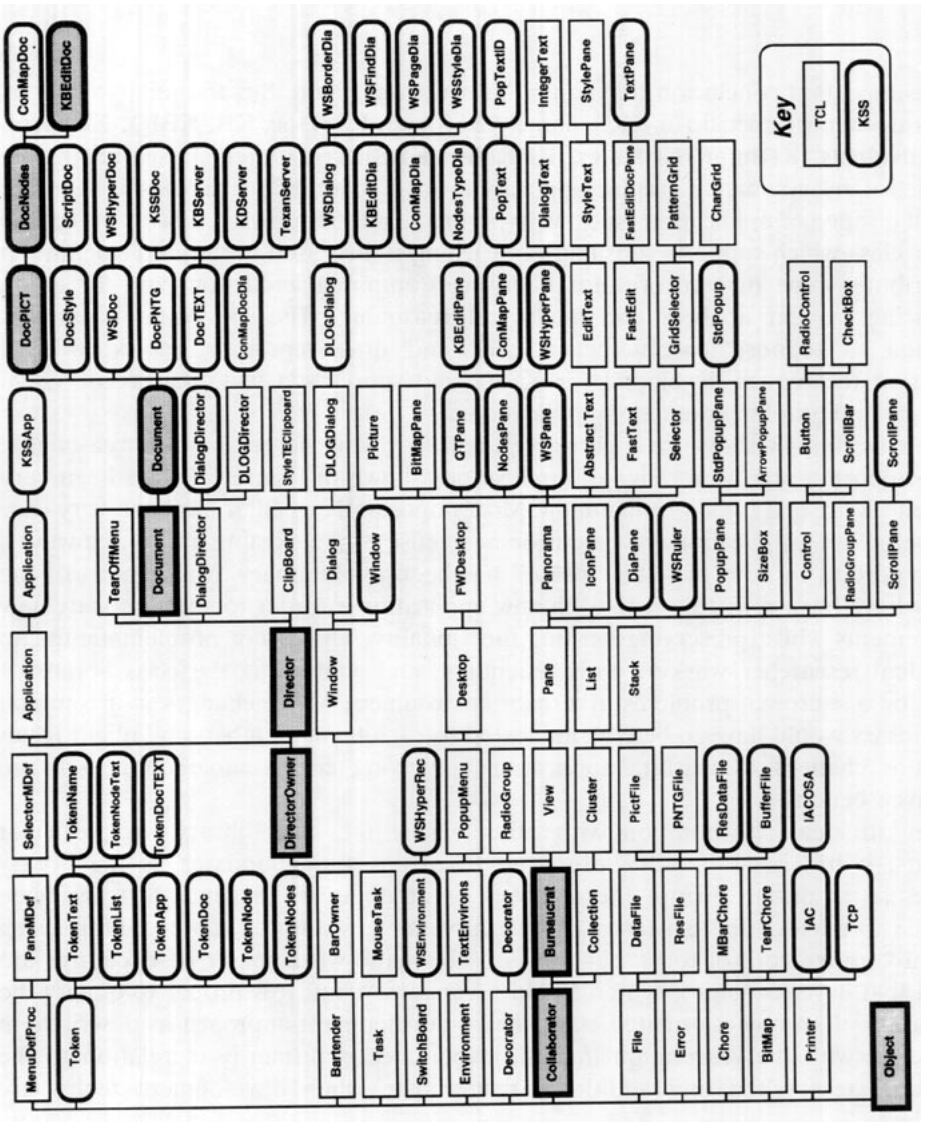

FIGURE 8. (Continued).

Document—the THINK class for managing a file and associated window
Document—override class for managing documents with extra features
DocPICT—managing a graphic file and associated window
DocNodes—managing an interactive graphic drawing file/window
KBEditDoc—specializes this for graphic editing of KRS knowledge structures

One important conclusion that can be drawn from Figure 8 is the relative paucity of classes that are specific to KRS, namely **KSSApp, KSSDoc, KB, KBIO, KBParse, KBIndividuals, KBInfer, KBRoles, KBRules, KBBitArr**. Other classes are highly generic to any interactive graphic application or to general data processing, and are used in a range of applications such as hypermedia and groupware. For example, the **BitArr** class which supports subsumption caching is also used in the implementation of attribute–value data records for high-speed empirical induction, and for graph processing in text analysis and analogical reasoning. The **DocNodes** class was designed to support the interactive click and drag repertory grid knowledge elicitation techniques developed for KSS0, and supports a wide variety of visual language and graphic interaction applications.

The class library shown in Figures 8(*a*) and (*b*), while large, is not unreasonably so in supporting the wide range of applications shown in Figure 1. It has been very simple to write and is easy to maintain. Most importantly of all, it has been very easy to develop new applications by extension and reuse of the existing library. However, it is important to note that this "reuse" has been evolutionary. Major parts of the library have been rewritten with each new application in order to accommodate new requirements whilst preserving existing functionality. This is not problematic for an individual researcher working with, amending, and controlling the class library. It would be a source of problems in a team environment where changes in a common class library would have to be communicated to each team member, and might result in major changes to existing applications involving considerable effort with no perceived benefit.

Even in local collaboration with others using the class library for particular projects, it has been found appropriate to avoid issuing updates as the library evolves so as not to disrupt the activities of other users. In general, it has been possible to manage the existence of several versions without excessive effort using standard source control tools, but this is problematic when major new features are added that have to be propagated to older versions. Thus, it is proper to qualify the evaluation of an object-oriented class library development approach as described in this paper with the reservation that the experience is primarily in relation to the individual researcher, and additional issues arise when development teams are considered.

It is also important to note that the class library design predated the availability in C++ of multiple inheritance supporting mixins, and of templates supporting parametric polymorphism (Waldo, 1993), and this restricted sub-classes to a tree and led to the duplication of classes differing only in basic data types. KRS is currently being reimplemented for C++ compilers that support multiple inheritance and templates, and the new class library structure is substantially more flexible, comprehensible and compact.

Theoretical basis of open-architecture KRS class library

One of the most interesting and important side-effects of the object-oriented class library implementation of a knowledge representation server was the theoretical insights that it generated into the nature of knowledge representation. As already noted, for purposes of practical knowledge acquisition it was essential to extend the CLASSIC representation model to include standard database data types such as numbers and dates. Initially this was done in an *ad hoc* fashion, but it became clear in interfacing new classes to the class library that there were general principles involved, and that the implementation of KRS could be restructured to support all the existing data types and provide an open architecture allowing new data types to be included simply and naturally. This was important to the future extension of KRS for use in different domains where new data types might be required. The reimplementation of CLASSIC's concepts, roles, individuals, extensional and cardinality constraints, and so on, however, led to a different theoretical model of KL-ONE-like knowledge representation systems than that common in the literature.

The most widely used framework for the formal analysis of knowledge representation has been the standard model and proof theory of first order logic. However, there are alternative algebraic models for first and higher order logics that are becoming increasingly used in programming language semantics because they represent abstract data types simply and naturally. For example, Boolean algebras that model propositional logic generalize to cylindric algebras (Henkin, Monk & Tarski, 1971) that model first order logic and have been used to analyse relational data bases (Imielinski & Lipski, 1984). Weaker algebraic models based on non-distributive lattices have been shown to give a comprehensive account of computational data types (Scott, 1976). An algebraic formalization of set theory without variables has been shown to provide adequate foundations for set theory and arithmetic (Tarski & Givant, 1987), and has been used to model description logics (Brink & Schmidt, 1992). A number of precise characterizations of algebraizable logics have been developed (Blok & Pigozzi, 1989). In the mid-1980s, Aït-Kaci (1984, 1986) gave a lattice-theoretic model of knowledge base languages with operational semantics through term rewriting that resolved many of the issues of complexity and deduction algorithms for term subsumption knowledge representation systems. This ψ-calculus is particularly interesting because it provides foundations for complex object representation in deductive databases, for type computation in functional programming languages, and for knowledge representation in artificial intelligence. An experimental programming language, LIFE (Aït-Kaci & Podelski, 1991), has been developed (Aït-Kaci, Meyer & van Roy, 1992) that combines the paradigms of logic programming, functional programming and object-oriented programming, and may be seen as a form of constraint logic programming.

The merits of algebraic, type-theoretic semantics for knowledge representation are that they provide formal models and complexity analyses that relate closely to the features and issues of existing knowledge representation systems. For example, they provide simple accounts of common constraints, such as cardinality, extensional inclusion and numeric ranges. Clearly, these are questions of naturality rather than logical power, since accounts of set theory and arithmetic can be developed in first

order logic, and algebraic feature constraint logics may be translated into first-order formulae (Smolka, 1992). However, natural models giving minimal formulations of representational requirements are valuable in both the software engineering of knowledge representation servers, and the effective presentation of the services offered to those using them.

The model for knowledge representation used in the open architecture implementation of the KRS class library is that the computational units are: for the T-box, *concepts*, identified with collections of *constraints* indexed by roles; and for the A-box, *individuals* represented as unique identifiers, each of which has an associated state whose value in any particular world is a concept. This has the normal semantics for KL-ONE systems that concepts are precisely a composition of constraints—changing the constraints changes the identity of the concept—whereas individuals are precisely identities of objects—changing the associated constraints changes the *state* of the individual, not its identity. In Zalta's (1988) terminology, abstract objects (concepts) *encode* properties whereas concrete objects (individuals) *exemplify* them.

This model makes the computational structure underlying theory and implementation an algebra of constraints, but leaves the nature of the constraints undefined. That is, from a theoretical perspective, any data type can be used in representation and deduction provided it can be modeled as a constraint algebra. From a practical perspective, this becomes the implementation objective—to support arbitrary families of constraints. In the KRS implementation, constraint data types and their operations are implemented as separate classes, and "plugging in" new data types involves adding a new class with four associated operations (composition, subsumption test, input and output). The kernel deduction systems for constraint propagation, for rule, inverse role and coreference application, and for model checking search, remain unchanged.

Constraint algebras for representation and inference

The semantics of constraints may be developed directly from informal requirements to a formal model. The key notions are that the *composition* of two constraints should be a well-defined constraint (binary function), that it makes no difference to compose a constraint with itself (idempotency), that grouping of multiple constraints in resolving them to binary compositions makes no difference (associativity), and that the order of application of constraints makes no difference (commutativity). Without these requirements, one would have the semantics of general *operators* rather than constraints. Together they imply that composition generates a *semi-lattice* in which it is the *join* operation. There is a natural order relation of *subsumption* of constraints, defined as one constraint subsumes another if their composition equals the second constraint. The semi-lattice can then be extended to be a full lattice by defining a dual, order-inverting meet operation [which, as a side effect, adds the lattice *adsorption* identities (Grätzer, 1971)]. A unique lowest element, or *zero*, can be defined in the lattice corresponding to the composition of incompatible constraints. A unique greatest element, or *unit*, can be defined corresponding to a universally applicable constraint.

Thus, the lattice structures common to all knowledge representation systems arise in general out of the basic primitive of a constraint. Any mathematical or logical

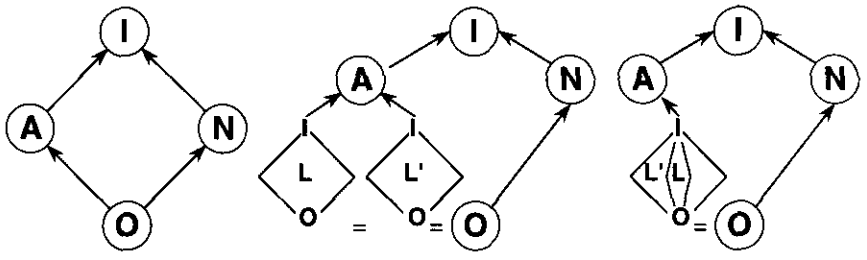

FIGURE 9. The basic constraint algebra and insertion of data types.

formulation is a representation of the properties of this primitive, and any representation schema with reasonably normal semantics will have a constraint algebra interpretation. Historically, this lattice-theoretic model of knowledge representation may be attributed (Simons, 1987) to Brentano's theory of judgment developed in his Würzburg logic letters in the early 1870s, and Simons (1982) has shown how Lesniewski's ontology (Luschei, 1962) arises naturally out of this approach. Birkhoff (1948) showed that the deductive closures of an arbitrary logic under the consequence operator form a complete lattice, and Wójcicki (1988) has developed an extensive theory of classical and non-classical logics based only on the properties of the consequence operator. Thus, lattice-theoretic algebraic semantics offer a complete alternative to the standard proof-theoretic and model-theoretic semantics of standard logics, and have the advantage that they extend naturally to the arbitrary constraint systems needed in real-world inference systems.

The detailed semantics of KRS and CLASSIC-like systems in terms of constraint algebras have been presented elsewhere (Gaines, 1993). Some concrete examples of the constraint algebras implemented in KRS are useful in understanding the principles involved. Figure 9 shows the basic constraint algebra on which KRS representation and inference is based, together with the way in which new data types are plugged in. The basic constraints are those shown in the four element lattice on the left of Figure 9 in which Ⓞ is the zero element, Ⓘ the unit element, and Ⓐ and Ⓝ are complementary elements distinct from them. The semantics of this lattice in terms of role constraints is:

Indeterminate	Ⓘ corresponds to a role possibly existing.
Overdeterminate	Ⓞ corresponds to a role existing but unfillable.
Applicable	Ⓐ corresponds to a role definitely existing.
Non-applicable	Ⓝ corresponds to a role definitely not existing.

This lattice supports no data types in itself but it can be extended to support any arbitrary combination of data types by inserting their constraint lattices between Ⓐ and Ⓞ such that their unit elements represent the relevant types and their zero elements are mapped to Ⓞ. Figure 9 center shows two mutually incompatible constraint lattices inserted in this way. Figure 9 right shows two compatible lattices inserted such that one is a sub-lattice of the other, corresponding to type coercion.

Set-theoretic constraint algebras

It has been shown elsewhere that the representation and inference capabilities of CLASSIC 1 may be implemented through a single set-theoretic data type that manages extensional and cardinality lower and upper bounds on sets of individuals (Gaines, 1993). Many other data types can be implemented by similar constraints on other set domains—for example point and interval integer, real and date types—and KRS provides a generic set constraint data type that encompasses all of these.

The general form of constraint on a sub-set X of a set S is defined by lower and upper bound sub-sets, L and U, and lower and upper bound measures, l and u, such that:

$$\varnothing \subseteq L \subseteq X \subseteq U \subseteq S \tag{1}$$

$$0 \leq l \leq \mathcal{M}X \leq u \leq \mathcal{M}S \tag{2}$$

where $\varnothing$ is the empty set and $\mathcal{M}$ is a mapping from a sub-set of S into a total order, a monotone *measure* (Oxtoby, 1971), such that the empty set maps to 0 and a larger set maps to a larger value in the order. The measure will typically be the cardinality of point sets and the total interval for interval sets. Its semantic role is to support constraints that limit sets by measure rather than content, for example number of members or total time spent. It is also significant in an open-world in enabling a set to be known to have become completely specified, or "closed" in description logic terminology, because it has reached the maximum measure specified.

Constraints defined by quadruples (L, U, l, u) form a natural constraint algebra relative to a set S under the rule of composition that, if $c = (L, U, l, u)$ and $d = (L', U', l', u')$, then the composition of c and d is:

$$c \sqcap d = (L \cup L', U \cap U', \max(l, l', \mathcal{M}(L \cup L')), \min(u, u', \mathcal{M}(U \cap U'))) \tag{3}$$

with the associated rules for the unit I and zero O of the constraint algebra:

$$x = (\{\}, S, 0, \mathcal{M}S) \Leftrightarrow x = I \tag{4}$$

$$x = (L, U, l, u), L \not\subseteq U \vee l > u \Leftrightarrow x = O \tag{5}$$

which allows it to be inserted into basic constraint algebra as an instance of Figure 9. For example, Figure 10 shows an example of a constraint algebra for the extensional and cardinality constraints on a set of two elements inserted in the base lattice of Figure 9.

The specification of the set upper bound U contains a flag which indicates whether the set or its complement is being specified. This allows sets to be specified negatively, such as "Not Fred or Mary". The unions and intersections of positive and negative sets and their combinations are well-defined and simple to compute. As already noted, this is significant for the open-world semantics of CLASSIC-like systems. It is also used to implement disjoint primitives as described later.

To provide the functionality of CLASSIC 1, KRS supports constraints over sets of individuals specified by unique identifiers, and constraints over sets of uniquely labeled items other than individuals, with the measure being the cardinality of the set in each case. The labeled items are used to support sets of arbitrary values of

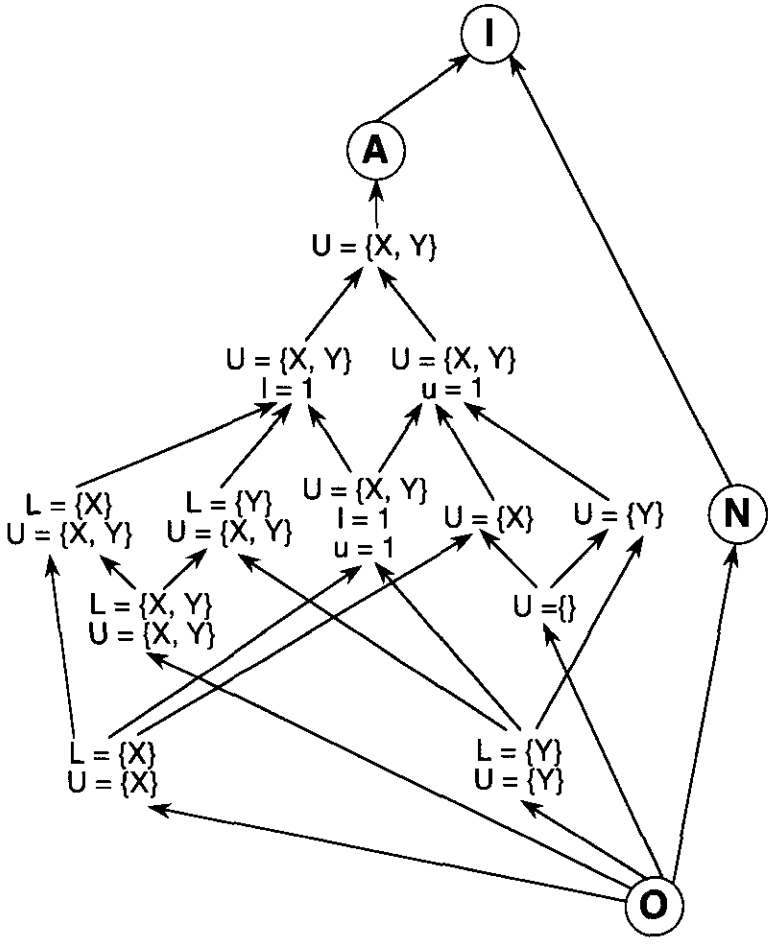

FIGURE 10. Example type lattice for set valued roles.

attributes, such as {red, green, blue}, that do not need to be represented as individuals with roles of their own. They are also used to implement primitive and disjoint primitive concepts as described later.

KRS also supports constraints over point sets of integers, reals and dates, and over interval sets of integers, reals and dates. The measure for point sets is the cardinality and for interval sets the total interval involved. The constraints are specified as sets of intervals. For example, a constraint on a point set of integers might be $(\{3\}, \{1, 2, 3, 4, 7\}, 1, 2)$ which can also be written as $(\{3\}, \{1 \leq 4, 7\}, 1, 2)$ and constrains a value to be either the integer 3, or two integers one of which is 3 and the other is 1, 2, 4 or 7. A constraint upon an interval set of dates might be $(\{\}, \{3\text{-Jan-}92 \leq 10\text{-Jan-}92, 5\text{-Feb-}92 \leq 9\text{-Mar-}92\}, 2, 5)$ which constrains a value to be between 2 and 5 days that are not necessarily consecutive in the periods 3-Jan to 10-Jan or 5-Feb to 9-Mar 1992.

The generic set constraints described encompass a wide range of requirements in their own right and are routinely available for all domains. More complex

constraints may be appropriate to specific domains, and this motivates the open architecture of KRS so that additional constraint algebras may be "plugged in" in a principled fashion as required.

Roles as indices to products of constraints

In terms of data structures, KL-ONE *roles* may be seen as corresponding to the fields of a record structure. Concepts are defined through constraints on the fields in a concept definition. Individual states are asserted as values (atomic constraints) of the fields in an individual state. Such record structures may be represented through a constraint algebra which is itself a product algebra of constraint algebras.

The construction is to take any constraint algebra, L, and consider an indefinite product of such algebras, $X = L \times L \times L \ldots$, indexed by a set of projections, Π, such that $\pi \in \Pi$, $x \in X \Rightarrow \pi x \in L$. There is a natural constraint lattice formed by X under the definitions:

$$x, y \in X, \pi \in \Pi \Rightarrow \pi(x \sqcap y) = \pi x \sqcap \pi y \tag{6}$$

$$\pi I = I \tag{7}$$

$$\pi O = O \tag{8}$$

That is, composition is done on a component by component basis. The base algebra L may be treated as part of X by defining a specific $\underline{\pi} \in \Pi$ and the mapping, $\mu: A \rightarrow X$:

$$a \in A \Rightarrow \pi\mu a = a \quad \text{if} \quad \pi = \underline{\pi}, \quad \text{else } \pi\mu a = I \tag{9}$$

The implementation follows the theory except that it conserves space in the usual way for record structures by omitting roles that are unconstrained (constrained by I) and explicitly listing the roles that are constrained. A concept record has the format shown in Figure 11.

The identity of a concept as an abstract object is determined by the constraint defined (Zalta, 1988). Hence the identifier has to be different for different concepts, and it is an error to give a second definition for the same identifier. Also, a concept defined with a different identifier but computed to define the same constraint is represented not by another concept record but by making the second identifier an alias for the first.

The "base constraint" corresponds to that for $\underline{\pi}$ in (9), and the role/concept pairs to those for the other projections in Π which have non-I constraints. Note that, if each concept is regarded as a node in a directed graph, then each role/concept pair may be regarded as defining an arc in the graph. Each arc is a directed link from concept node to concept node and is labeled by the role involved. It is this "concept definition graph" that is the basis of theoretical analyses of subsumption computations and the correctness and completeness of reasoning in description logics (Nebel, 1990; Borgida & Patel-Shneider, 1994). Its direct implementation as a "visual language" also provides a natural interface for specifying, querying and receiving results from description logics (Gaines, 1991*b*).

An individual record has the format shown in Figure 12. The identity of an individual as a concrete object is determined by its unique identifier (Zalta, 1988).

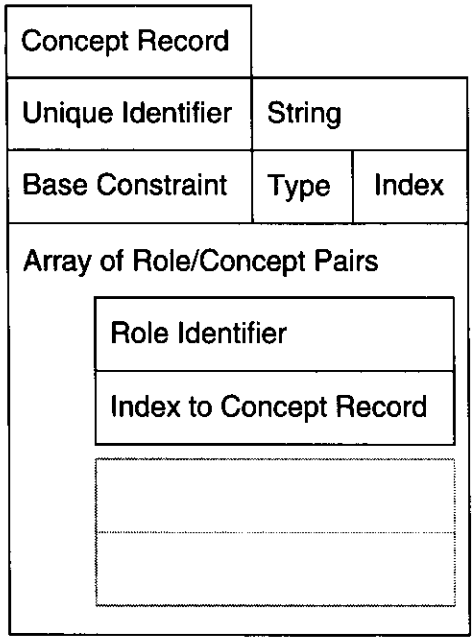

FIGURE 11. Structure of a concept definition record.

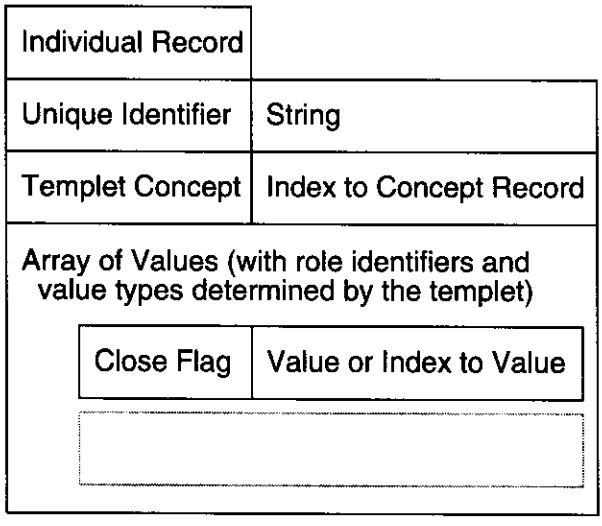

FIGURE 12. Structure of an individual record.

Hence, different assertions about individuals with the same identifier are treated as assertions about the same individual and composed to give the individual's state. KRS also supports individuals (Skolem individuals) where the identifier is not yet known, for example "a person" rather than "Fred Smith". These arise naturally through existential quantifiers represented as lower bound cardinality constraints.

KRS follows CLASSIC in distinguishing conceptual assertions about an individual which are composed into a "templet concept", and value assertions which are composed into an array of values. The templet concept is also used to specify the relevant roles and the types of the values in this array. The term is chosen to correspond with Kelly's (1955) constructivism "Man looks at his world through transparent templets which he creates and then attempts to fit over the realities of which the world is composed", and the development of knowledge representation in terms of personal construct psychology (Gaines & Shaw, 1993*a*). Since many role fillers are single-valued, to optimize space utilization a coding scheme is used whereby a value consisting of a set with zero or one member may be packed directly into the array of values, but sets with two or more members are referenced through an index into a separate array of sets. The "close flag" is used to indicate whether a set value has been asserted to be fully specified (closed) or whether it is still open to the addition of more members.

Since the values may themselves be sets of individuals, an individual record also has a natural representation as a graph structure which is somewhat more complex than that for a concept definition. The "recognition" of an individual as being an instance of a concept corresponds to determining whether the individual graph is an instance of the concept graph, and the algorithm is essentially one of graph matching. KRS follows CLASSIC in not taking account of individual states when computing subsumption relations between concepts. However, it takes full account of these states when computing whether an individual is an instance of a concept. This involves careful consideration of whether value sets are fully specified. For example, Fred with friends John and Mary both having red hair would be recognized as an instance of an individual all of whose friends have red hair only if it had been asserted that the set of Fred's friends was closed, or if this could be computed from some other consideration, such as Fred having a maximum of two friends.

The instance recognition algorithm in KRS is quite separate from and rather more complex than the concept subsumption algorithm because it is targeted to be complete in reasoning about cases in order to ensure that individual classification and rule firing are logically complete. Concept subsumptions are computed and cached, and used to speed up instance recognition. However, the impact of concept subsumption being incomplete in most applications of KRS is that reasoning about individuals is slower than it might be. Indeed, if one turns off concept subsumption completely the only impact in the "expert system" types of application for which KRS is designed is that reasoning slows down. This corresponds to Vilain's (1991) rationale for the utility of KL-ONE-like systems, despite problems of intractability, being that practical uses involve reasoning about finite worlds of individuals, and Doyle and Patil's (1991) critique of the limited value of subsumption computation in description logics. If the applications primarily involve concrete cases rather than general theorem proving then the critical algorithms are those concerned with the recognition of instances rather than the subsumption of concepts.

Implementation of classic capabilities in KRS

The descriptions in the previous sections indicate how some of the eight semantic structures of CLASSIC are represented in the KRS implementation. Concepts, roles and individuals are represented through the record structures described in the previous section. Cardinality and explicit extensional concepts are represented through the set constraints described in the section prior to that. This section describes how the other semantic structures are implemented. It also gives some concrete examples of KRS record structures corresponding to specific concept definitions and individual assertions.

Primitive concepts and disjoint constraints on them are represented through a hidden role whose fillers are the names of the primitive concepts, and where the disjoint constraint is a negative extensional set constraint. For example, defining a primitive concept "Animate" and a concept "Inanimate" disjoint to it, both of type "Individual" (as shown at the top of Figure 6), results in the concept record structures shown in Figure 13.

Concepts 1 and 3 are generated as "anonymous" concepts to hold the set-theoretic constraints upon the hidden role "Primitive" in the defined concepts. Concept 2 is a simple primitive definition of a concept "Animate" whose base constraint is that it applies to sets of individuals but is otherwise unconstrained by extensional or cardinality constraints. It is made primitive by concept 1 being specified as a constraint upon its "Primitive" role, that is that the set filling this role must contain the label "Animate". Concept 4 is a disjoint primitive definition of a concept "Inanimate" whose base constraint is that it applies to sets of individuals but is otherwise unconstrained by extensional or cardinality constraints. It is made primitive and disjoint from "Inanimate" by concept 3 being specified as a constraint upon its "Primitive" role, that is that the set filling this role must contain the label "Animate" and cannot contain the label "Inanimate". This ensures that the composition of concepts 3 and 4 is incoherent. Note the asymmetry in the concept definition which corresponds to concept 2 being defined first and concept 4, disjoint to it, being defined in terms of it. This corresponds to the avoidance of recursive definitions in KL-ONE-like systems. It also allows further concepts disjoint to "Animate" or "Inanimate" to be defined later without involving changes to the existing concept definitions.

The use of a hidden role for primitives is ontologically significant since it is often appropriate to represent natural kinds by an explicit role with labels as fillers having a complex pattern of disjoint relations that might correspond to a rating scale (Gaines & Shaw, 1993*a*). What are initially represented as simplistic binary distinctions, such as *male–female*, often turn out to have a richly differentiated ontological structure (Lorber, 1994).

Implicit extensional concepts, or coreference constraints, are different from the constraints defined so far because they involve equality constraints between different role fillers. For example, that the set of friends of Fred is the same as the set of friends of Fred's wife. Such constraints are implemented in KRS by attaching records of equivalent role chains to concepts as part of the concept definitions. The equivalence of the role chains is taken into account by merging the constraints specified at the end of each role chain, and taking role chain equivalences into

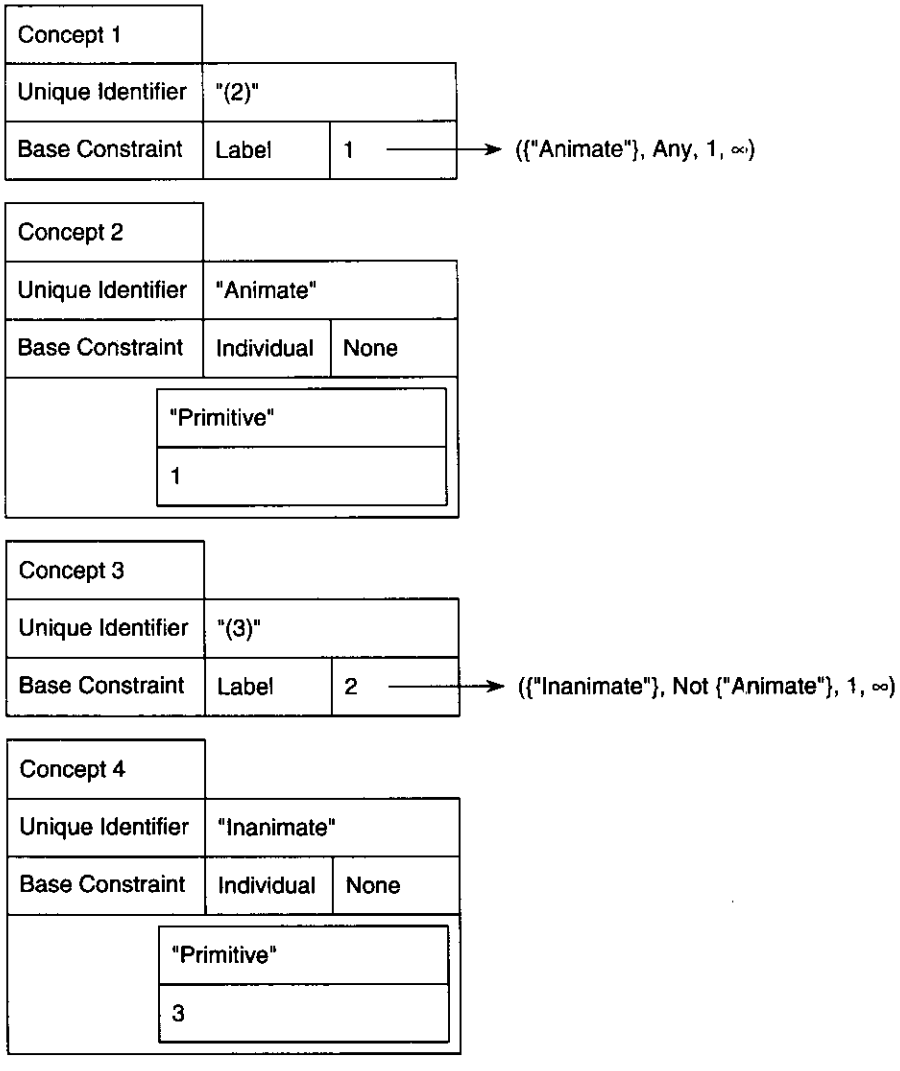

FIGURE 13. Concept records generated by a pair of disjoint primitive definitions.

account in subsumption computation. When implicit extensional concepts are asserted to apply to individuals, the fillers of the equivalent roles are merged and a single index to the merged set placed as a value in the equivalent roles in the individual. KRS does not allow coreference constraints to apply between two points in the same role chain since this leads to the same problems as recursive definitions; CLASSIC 2 does allow such constraints (Borgida & Patel-Shneider, 1994).

Rules are represented in KRS as a pair of concepts such that if the first concept applies to an individual then the second one is asserted to apply to that individual. A list of the rules that should be triggered by a concept is stored as part of the concept record. KRS also supports rules with exceptions such that one rule being applicable prevents another rule being applicable (Gaines, 1991*a*). The proper implementation

of this involves a careful distinction between an individual being an instance of a concept, being definitely not an instance, or it being open that the individual may with further assertions become either an instance or a non-instance. Reports on whether rules apply, do not apply, or are open, are very useful in explaining the reasoning of KRS to users or in requesting additional information in order to clarify whether open rules apply. The support of default reasoning is another important extension required of description logics (Baader & Hollunder, 1992) and rules with exceptions may be used to provide normal defaults (Reiter, 1980) by assuming that open rules will not fire. However, a prioritization of defaults (Quantz & Royer, 1992; Giordano & Martelli, 1994) is also required since, otherwise, the rules that are allowed to fire as a result of their exceptions not firing may lead to an incoherent state of the knowledge base.

A concrete example of KRS data structures

To provide a concrete example of KRS data structures, Figure 14 is a representative knowledge structure represented in the visual language of KDraw.

At the top left, an "employee" is defined to be a primitive concept with a role "salary" whose filler is constrained to be an integer in the range 15 000–160 000 and a primitive concept "US $" (note that basic data types such as integers can be given more specific semantics in this way), and with a role "division" constrained to be a "division".

Below "employee", a "foreman" is defined to be a primitive concept inheriting the properties of "employee" and having the additional constraint on the role "salary" that it is in the range 35 000–70 000, and the additional constraint on the role "division" that it is single-valued and its role "classification" contains the label "manufacturing".

At the top right, a "division" is defined to be a primitive concept with a role "revenues" whose filler is constrained to be an integer and a primitive concept "US $000", and with a role "classification" constrained to be one label from "sales", "marketing", etc.

At the center left, a rule "senior employee" is defined whose premise is the concept "senior foreman" defined to be a "foreman" with a "salary" of at least 45 000 and a "division" whose "revenues" are at least 1000. The conclusion is that any individual recognized to be an instance of "senior employee" will be asserted to be a "senior employee" with at least 1 "assistant" who will be asserted to be a "senior employee's assistant".

At the bottom left, the individual "Fred Smith" is asserted to be a "foreman" with "salary" 50 000, "division" role filled by the individual "Body Works" and "assistant role" filled by "Sam Jones". At the bottom right, the individual "Body Works" is asserted to have "revenues" of 3000.

When the "Compile" button in Figure 14 is clicked, the code shown in Figure 15 is generated which represents the same knowledge structures in a textual description language. In practice, users do not see this code because expressions in the visual language can be selected, copied and pasted directly into the knowledge representation server.

If the knowledge structure of Figure 14 or 15 is loaded into KRS then the "senior

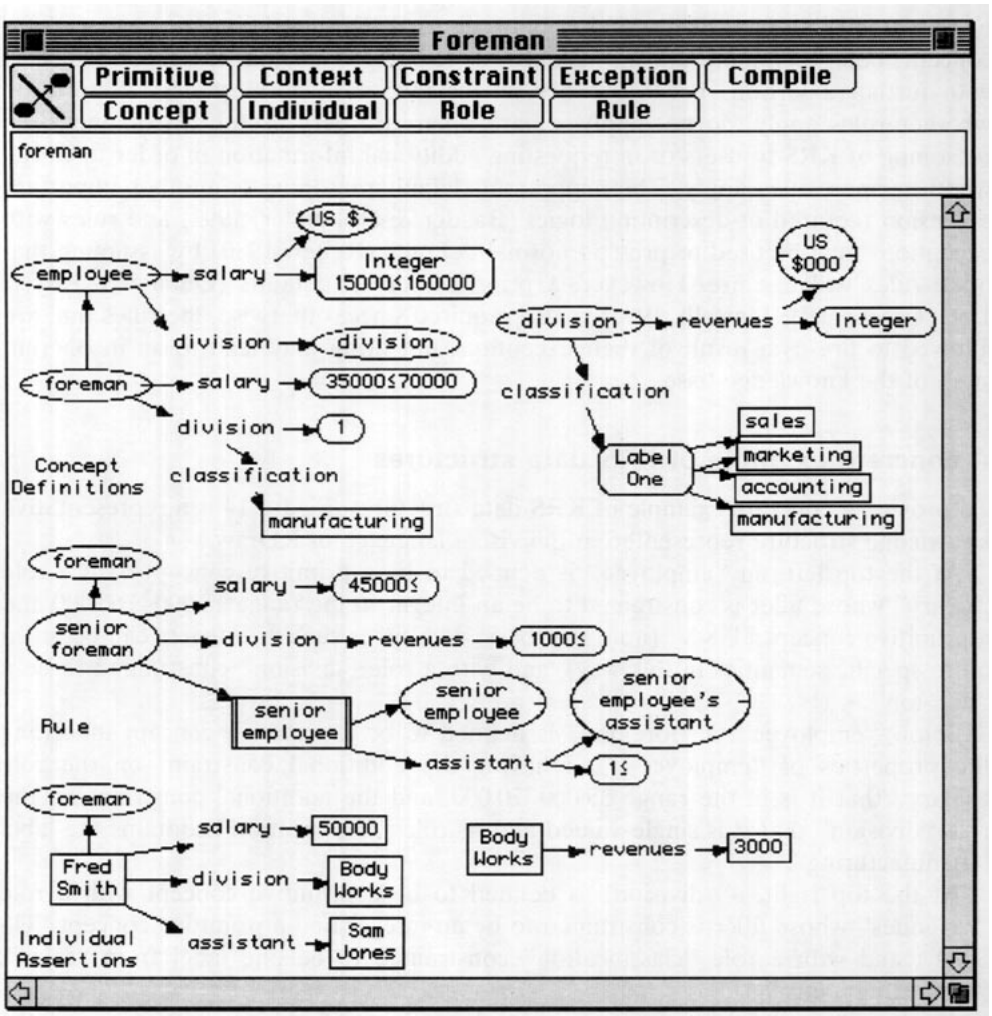

FIGURE 14. Visual representation of a knowledge structure in KDraw.

employee" rule fires for "Fred Smith" and he is asserted to be a "senior employee" and his assistant "Sam Jones" is asserted to be a "senior employee's assistant". Figure 16 shows a listing from the server of all the concept records generated by this knowledge structure. The "anonymous" concepts generated by KRS rather than defined by the user are identified by the record number in parentheses.

As an explanation of a typical record, record 14 corresponds to the definition of "division" in the knowledge structure. Its type is "Open" because no particular type was specified. An "Open" type will become a specific type when the concept is asserted to apply to an individual. The "Primitive" role is constrained by concept record 13 which specifies a set of type "Label" including "division". The "classification" role is constrained by concept record 11 which specifies a set of type "Label" with exactly one member in the set {sales, marketing, accounting, manufacturing}. The "revenues" role is constrained by concept record 12 which

```
Primitive(US $)
Primitive(US $000)
Primitive(senior employee)
Primitive(senior employee's assistant)
Primitive(division
  (All classification, (Label One sales, marketing, accounting, manufacturing))
  (All revenues, US $000, (Integer))
)
Primitive(employee
  (All salary, US $, (Integer 15000≤160000))
  (All division, division)
)
Primitive(foreman, employee
  (All salary, (35000≤70000))
  (All division, (1), (All classification, (Include manufacturing)))
)
Concept(senior foreman, foreman
  (All salary, (45000≤))
  (All division, (All revenues, (1000≤)))
  (Rule senior employee)
)
Rule(senior employee, senior employee
  (All assistant, senior employee's assistant, (1≤))
)
Individual(Fred Smith, foreman
  (Fills salary, 50000)
  (Fills division, Body Works)
  (Fills assistant, Sam Jones)
)
Individual(Body Works
  (Fills revenues, 3000)
)
```

FIGURE 15. Textual representation of the knowledge structure compiled from Figure 14.

specifies an Integer with its role "Primitive" constrained by concept record 5, that is, including a label "US $000", meaning the unit is 1000s of dollars.

Figure 17 shows a listing from the server of all the individual records generated by this knowledge structure. The first concept record specified in the Templet field is that prior to any rules firing, and the second is that applying after inference is complete. Thus, "Fred Smith" was initially an instance of concept record 35 and became an instance of concept record 41. From Figure 16 it can be seen that concept record 35 specifies that "Fred Smith" is an instance of the primitive concepts "employee" and "foreman", has a "salary" which is an Integer in the range 35 000–70 000, and so on. Concept record 41 updates this to specify that he is an instance of "senior employee" also, and his "assistant" is an instance of "senior employee's assistant".

The asserted values filling other roles are shown without parentheses, and the

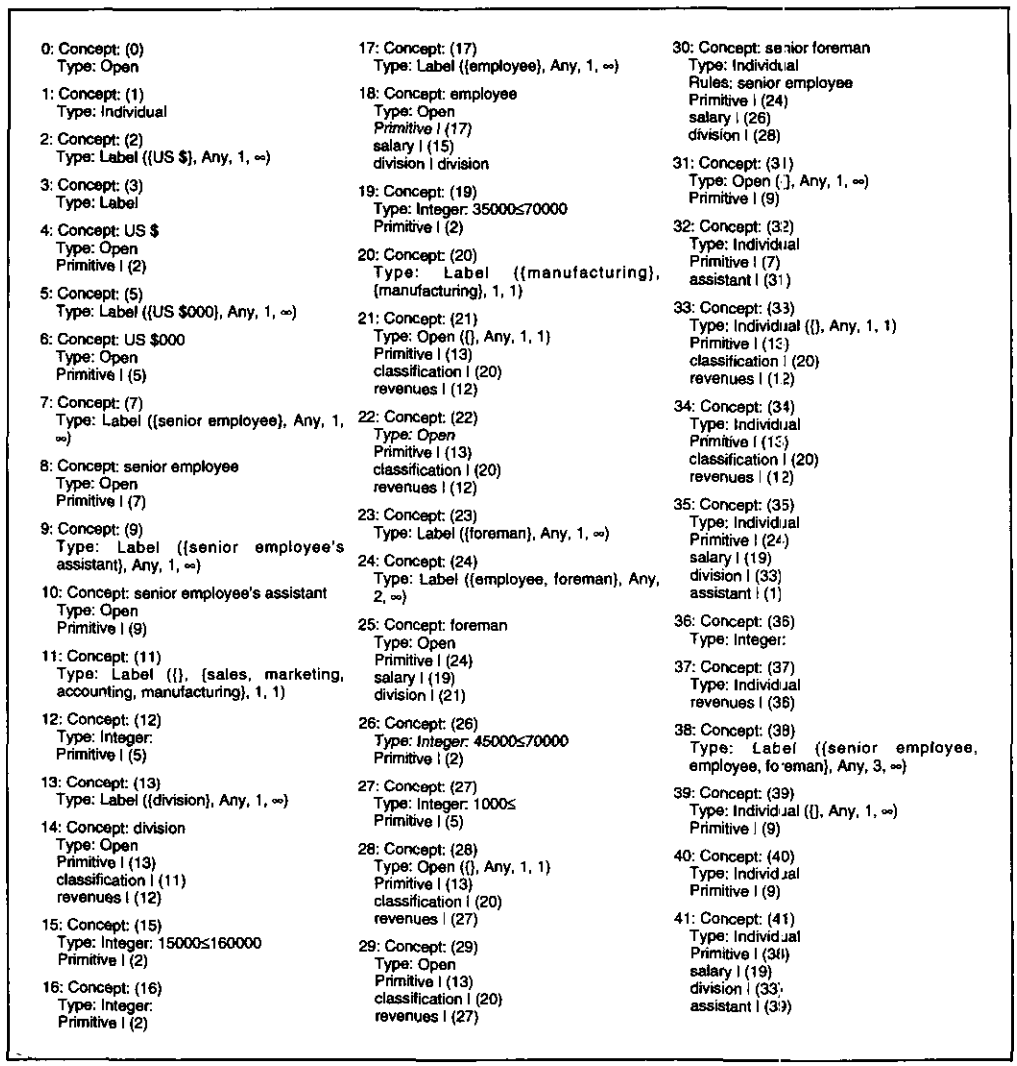

FIGURE 16. Concept records generated for knowledge structure of Figures 14 and 15.

inferred ones in parentheses. This corresponds to KRS keeping track of the distinction between what is asserted and what is inferred. Figure 18 illustrates this in the output graph that KRS generates to show the state of the individuals in the knowledge base after inference. The inferences made are clearly visible. For example, it has been inferred that all fillers of the assistant role of "Fred Smith" must be classified as "senior employee's assistant", and, in particular, that "Sam Jones" must be thus classified. The output grapher links related individuals and hence tends to make the relationship clearer than is the case with input graphs, which may be entered piecemeal.

```
0: Individual: Fred Smith
   Templet: (35), (41)
   Primitive = (senior employee, employee, foreman)
   salary = 50000
   division = Body Works
   assistant = Sam Jones
1: Individual: Body Works
   Templet: (37), (34)
   Primitive = (division)
   classification = (manufacturing)
   revenues = 3000
2: Individual: Sam Jones
   Templet: (1), (40)
   Primitive = (senior employee's assistant)
```

FIGURE 17. Individual records generated for knowledge structure of Figures 14 and 15.

Plug-in data types in KRS

The relation between the inference mechanism and the plug-in data types of KRS is shown in Figure 19. The T-box and the A-box support the normal features of KL-ONE-like systems in allowing concepts to be defined, and assertions to be made about individuals in terms of these concepts. What is essentially type propagation inference can then be used to deduce the consequences of the assertions through

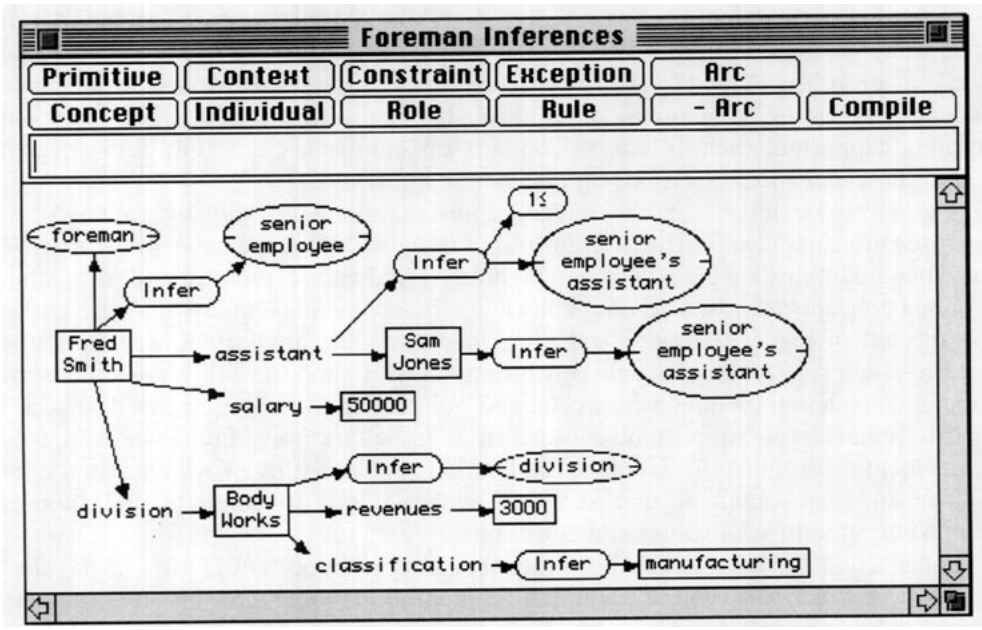

FIGURE 18. States of individuals graphed by KRS.

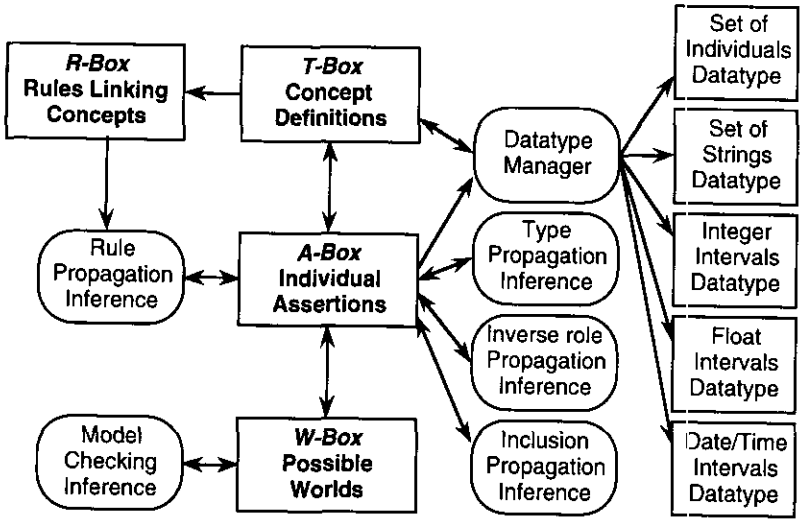

FIGURE 19. Plug-in data types in relation to the KRS inference mechanisms.

reference to the definitions. The R-box supports the rule schema of CLASSIC, extended to handle rules with exceptions, such that an individual recognized as satisfying one concept has another automatically asserted of it. This integrates production rule and frame-based reasoning. The W-box supports model checking (Halpern & Vardi, 1991) or "puzzle mode" reasoning in which, if necessary, after propagation of type and rule constraints a search of possible worlds is carried out to determine whether additional conclusions can be drawn because not to do so would lead to absurdity. A wide variety of data types is supported through a data type manager accessing separate modules through a uniform interface such that constraints supported, such as interval bounds on numbers, enter fully into concept definitions, individual assertions and deductive inference.

The algebraic model of knowledge representation leads directly to the open architecture class library structure illustrated in Figure 20. The knowledge base class has two main instance variables, one holding individual records and the other holding concept records. The base knowledge base class implements the type lattice on the left of Figure 9, having codes for indeterminate, applicable, non-applicable and overdeterminate values and constraints. It also supports other constraints by reference codes containing a type code and pointer. The type code is used to access a list of data type support objects which is initially empty. For each data type implemented, an object is added to the list that supports it by providing storage for values and constraints of that sub-type, and methods to compute and support input/output with such values and constraints.

Figure 20 at the bottom left also illustrates another important feature of the class library construction. Type subsumption computation for record structures is very simple when expressed recursively. However, to attain the known complexity lower bound requires that already computed subsumption relations be cached to avoid duplicate computation. This caching is implemented in a sub-class in KRS, allowing

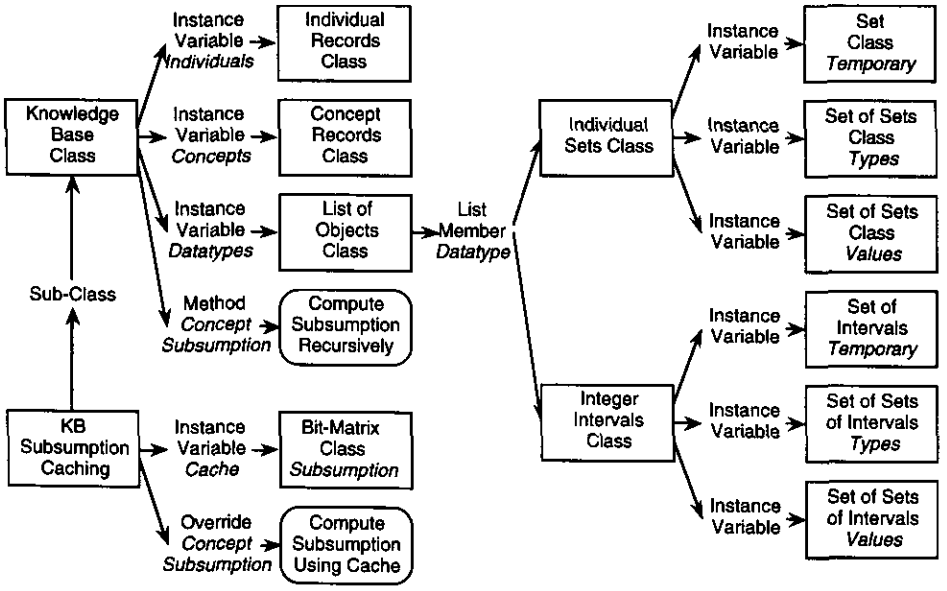

FIGURE 20. Plug-in data types in relation to the knowledge base class library structure.

the simple recursive computation to be called as a sub-class method during debugging as a check on the correctness of the caching algorithm.

When examined in relationship to the algebraic semantics described previously, Figures 19 and 20 show the close relationship between the class library implementation and the theoretical foundations of the knowledge representation server. The basic constraint algebra of Figure 9 is sub-classed to support various set-based constraint algebras that are extensions of it. However, the additional inference mechanisms required are encapsulated within the sub-class objects so that the overall inference schema shown in Figure 19 remains unchanged.

It is reasonable to expect that a constraint-theoretic architecture for a KL-ONE-like description logic provides a basis for the implementation of systems that incorporate recent developments in constraint logic programming (Benhamou & Colmerauer, 1993) and general representation and inference based on constraint satisfaction (Tsang, 1993). The constraint satisfaction literature now subsumes so much information processing—numerical and logical—within a well-principled framework that it appears to offer the strong possibility of a unified foundation for all aspects of representation and inference.

It is also to be expected that the graph-theoretic foundations of algorithms for concept subsumption and instance recognition in general description logics will provide a basis for the implementation of systems that incorporate recent developments in graph-based representation and inference (Ellis & Levinson, 1994). These developments already underlie the most promising high-speed and yet richly-featured description logic implementations such as PEIRCE (Ellis & Levinson, 1992) and KRIS (Baader, Hollunder, Nebel & Franconi, 1992).

Conclusions

Object-oriented class libraries offer the potential for individual researchers to manage the large bodies of code generated in the experimental development of complex interactive systems. This article has analysed the structure of such a class library that supports the rapid prototyping of a wide range of knowledge support systems including collaborative networking, shared documents, hypermedia, machine learning, knowledge acquisition and knowledge representation, and various combinations of these technologies. The overall systems architecture has been presented in terms of a heterogeneous collection of systems providing a wide range of application functionalities. The use of the class library to develop complex interactive and knowledge-based systems has been exemplified through applications to group writing, multimedia and knowledge-based systems.

In terms of the theme of this special issue, the implementation of KRS is particularly interesting because KL-ONE-like representation schema may be regarded as themselves object-oriented, with concepts corresponding to classes, and individuals corresponding to objects. How does the object-oriented implementation interact with the object-oriented application? From one perspective, as already noted in the literature (Troyer, Keustermans & Meersman, 1986), the object-oriented implementation contributes nothing directly to the object-oriented knowledge representation system being implemented. That is, the C++ classes are not in themselves the KRS conceptual structures. This is to be expected because the KRS conceptual structures are dynamic at run-time whereas the C++ class structures are static. From this perspective, 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 independent in their object-oriented functionality, but the lower level is pragmatically highly important to the design and implementation of the upper level.

However, as analysed in the latter part of the article, there are deep theoretical relations between the algebraic model of the knowledge representation system and its class library implementation. It has been shown that modeling the server through intensional algebraic semantics leads naturally to an open-architecture class library into which new data types may be plugged as required without change to the basic deductive engine. This may still be viewed as a software engineering feature of the implementation, but there is more to it than that. The data types in KRS are being reimplemented through common classes that are specialized through parametric polymorphism using templates (Stroustrup, 1991). In this construction the C++ features are closely identified with the algebraic semantics of the knowledge representation system. This is consistent with the evolution of C++ which has continually moved towards higher-level representation features (Waldo, 1993). If C++ types were dynamic at run-time then even more of the functionality of KRS could be identified with that of its implementation language.

The experience reported in this article shows that the development of a principled class library targeted on complex interactive applications does empower the individual researcher in the rapid prototyping of experimental systems. However, as

already noted, much of the power of the approach stems from the cumulative evolution of the class library through successive applications. The class library is itself a vehicle for recording and operationalizing the increasing knowledge and experience of its primary user, rather than a static foundation for development. This implies that the positive results may not generalize to team projects where greater rigidity is required in the class library in order to facilitate project management and inter-member coordination.

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada. I am grateful to Ron Brachman and his colleagues at AT&T for access to CLASSIC manuals and related research material. I am grateful to Mildred Shaw and to the anonymous referees for detailed suggestions that greatly improved the presentation of the KRS-related material, and to Hermann Kaindl for his support and encouragement in the preparation of this paper.

References

ABRETT, G. & BURSTEIN, M. H. (1988). The KREME knowledge editing environment. In J. H. BOOSE & B. R. GAINES, Eds. *Knowledge Acquisition Tools for Expert Systems*, pp. 1–24. London: Academic Press.

AÏT-KACI, H. (1984). *A lattice-theoretic approach to computation based on a calculus of partially-ordered types*. PhD. Theses, University of Pennsylvania, Philadelphia, PA, USA.

AÏT-KACI, H. (1986). An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, **45**, 293–351.

AÏT-KACI, H., MEYER, R. & VAN ROY, P. (1992). *Wild LIFE: a user manual*. Digital Equipment Corporation Research Laboratory, Paris, France.

AÏT-KACI, H. & PODELSKI, A. (1991). *Towards a meaning of LIFE*. Report PRL-RR-11, Digital Equipment Corporation Research Laboratory, Paris, France.

APPLE (1993*a*). *Inside Macintosh Interapplication Communication*. Reading, MA: Addison-Wesley.

APPLE (1993*b*). *OpenDoc Developer Overview*. Cupertino, CA: Apple Corporation.

BAADER, F. & HANSCHKE, P. (1991). A scheme for integrating concrete domains into concept languages. *IJCAI'91: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 452–457. Sydney: Morgan Kaufmann.

BAADER, F. & HOLLUNDER, B. (1991). KRIS: Knowledge representation and inference system. *ACM SIGART Bulletin*, **2**(3), 8–14.

BAADER, F. & HOLLUNDER, B. (1992). Embedding defaults into terminological knowledge representation formalisms. *Proceedings of KR'92: Third International Conference on Principles of Knowledge Representation and Reasoning*, pp. 306–317. San Mateo, CA: Morgan Kaufmann.

BAADER, F., HOLLUNDER, B., NEBEL, B. & FRANCONI, E. (1992). An empirical analysis of optimization techniques for terminological representation systems or: making KRIS get a move on. *Proceedings of KR'92: Third International Conference on Principles of Knowledge Representation and Reasoning*, pp. 270–281. San Mateo, CA: Morgan Kaufmann.

BARRETT, E., Ed. (1989). *The Society of Text: Hypertext, Hypermedia and the Social Construction of Information*. Cambridge, MA: MIT Press.

BASS, L. & DEWAN, P., Eds (1993). *User Interface Software*. Chichester, UK: John Wiley.

BENHAMOU, F. & COLMERAUER, A., Eds (1993). *Constraint Logic Programming*. Cambridge, MA: MIT Press.

BIGGERSTAFF, T. J. & PERLIS, A. J., Eds (1989*a*). *Software Reusability Volume I: Concepts and Models*. Reading, MA: Addison-Wesley.

BIGGERSTAFF, T. J. & PERLIS, A. J., Eds (1989*b*). *Software Reusability Volume II: Applications and Experience*. Reading, MA: Addison-Wesley.

BIRKHOFF, G. (1948). *Lattice Theory*. Providence, RI: American Mathematical Society.

BLOK, W. J. & PIGOZZI, D. (1989). *Algebraizable Logics*. Providence, RI: American Mathematical Society.

BOOCH, G. (1991). *Object Oriented Design with Applications*. Redwood City, CA: Benjamin/Cummings.

BOOSE, J. H. (1984). Personal construct theory and the transfer of human expertise. *Proceedings AAAI-84*, pp. 27–33. San Mateo, CA: American Association for Artificial Intelligence.

BOOSE, J. H. & BRADSHAW, J. M. (1987). Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. *International Journal of Man-Machine Studies*, **26**, 3–28.

BOOSE, J. H., BRADSHAW, J. M., KITTO, C. M. & SHEMA, D. B. (1989). From ETS to Aquinas: six years of knowledge acquisition tool development. In J. H. BOOSE, B. R. GAINES & J. G. GANASCIA, Eds. *Proceedings of EKAW-89: Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, pp. 502–516. Paris: University of Paris Press.

BOOSE, J. H. & GAINES, B. R. (1988). *Knowledge Acquisition Tools for Expert Systems*. London: Academic Press.

BORGIDA, A., BRACHMAN, R. J., MCGUINESS, D. L. & RESNICK, L. A. (1989). CLASSIC: a structural data model for objects. *Proceedings of 1989 SIGMOD Conference on the Management of Data*, pp. 58–67. New York: ACM Press.

BORGIDA, A. & PATEL-SHNEIDER, P. F. (1994). A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, **1**, 277–308.

BRACHMAN, R. J. (1977). What's in a concept: structural foundations for semantic nets. *International Journal of Man-Machine Studies*, **9**, 127–152.

BRACHMAN, R. J., GILBERT, V. P. & LEVESQUE, H. J. (1985). An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. *Proceedings of IJCAI 85*, pp. 547–551. San Mateo, CA: Morgan Kaufmann.

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, CA: Morgan Kaufmann.

BRACHMAN, R. J., MCGUINESS, D. L., PATEL-SCHNEIDER, P. F., RESNICK, L. A. & BORGIDA, A. (1991). Living with Classic: when and how to use a KL-ONE-like language. In J. F. SOWA, Ed. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pp. 401–456. San Mateo, CA: Morgan Kaufmann.

BRACHMAN, R. J. & SCHMOLZE, J. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, **9**(2), 171–216.

BRINK, C. & SCHMIDT, R. A. (1992). Subsumption computed algebraically. In F. LEHMANN, Ed. *Semantic Networks in Artificial Intelligence*, pp. 329–342. Oxford: Pergamon Press.

CERCONE, N. & MCCALLA, G., Eds (1987). *The Knowledge Frontier: Essays in the Representation of Knowledge*. New York: Springer.

COMPTON, P., EDWARDS, G., KANG, B., LAZARUS, L., MALOR, R., PRESTON, P. & SRINIVASAN, A. (1992). Ripple down rules: turning knowledge acquisition into knowledge maintenance. *AI in Medicine*, **4**(6), 463–475.

CRANDALL, G. (1990). *Word Solution Engine Programmer's Manual*. Vancouver: DataPak.

CULLINGFORD, R. E. (1986). *Natural Language Processing: A Knowledge Engineering Approach*. Totowa, NJ: Rowman and Littlefield.

DEVANBU, P., SELFRIDGE, P. G., BALLARD, B. W. & BRACHMAN, R. J. (1989). A knowledge-based software information system. *IJCAI'89: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 110–115. San Mateo, CA: Morgan Kaufmann.

DOYLE, J. & PATIL, R. S. (1991). Two theses of knowledge representation: language

restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, **48**(3), 261–297.

DRUCKER, D. L. & MURIE, M. D. (1992). *QuickTime Handbook*. Carmel, IN: Hayden.

ELLIS, G. & LEVINSON, R., Eds (1992). *Proceedings of the First International Workshop on PEIRCE: A Conceptual Graphs Workbench*. University of Queensland, Australia. ftp.cs.uq.oz.au /pub /TECHREPORTS /department /TR0241.ps.Z.

ELLIS, G., LEVINSON, R. & ROBINSON, P. J. (1994). Managing complex objects in PEIRCE. *International Journal of Human–Computer Studies*, **41**, 109–148.

FILMAN, R. E. (1988). Reasoning with worlds and truth maintenance in a knowledge-based system shell. *Communications of the ACM*, **31**(4), 382–401.

FININ, T., WEBER, J., WIEDERHOLD, G., GENESERETH, M., FRITZSON, R., MCKAY, D., MCGUIRE, J., SHAPIRO, S. & BECK, C. (1992). *Specification of the KQML agent-communication language*. The DARPA Knowledge Sharing Initiative External Interfaces Working Group.

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, CA: Morgan Kaufmann.

GAINES, B. R. (1990*a*). Knowledge representation servers: a generic technology for knowledge acquisition and knowledge-based systems. In H. MOTODA, R. MIZOGUCHI, J. BOOSE & B. R. G. GAINES, Eds. *Knowledge Acquisition for Knowledge-Based Systems*, pp. 413–430. Tokyo: Ohmsha.

GAINES, B. R. (1990*b*). Knowledge support systems. *Knowledge-Based Systems*, **3**(3), 192–203.

GAINES, B. R. (1991*a*). Integrating rules in term subsumption knowledge representation servers. *AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 458–463. Menlo Park, CA: AAAI Press/MIT Press.

GAINES, B. R. (1991*b*). An interactive visual language for term subsumption visual languages. *IJCAI'91: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 817–823. San Mateo, CA: Morgan Kaufmann.

GAINES, B. R. (1993). A class library implementation of a principled open architecture knowledge representation server with plug-in data types. *IJCAI'93: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 504–509. San Mateo, CA: Morgan Kaufmann.

GAINES, B. R. (1994). A situated classification solution of a resource allocation task represented in a visual language. *International Journal of Human–Computer Studies*, **40**(2), 243–271.

GAINES, B. R. & BOOSE, J. H. (1988). *Knowledge Acquisition for Knowledge-Based Systems*. London: Academic Press.

GAINES, B. R. & COMPTON, P. (1993). Induction of meta-knowledge about knowledge discovery. *IEEE Transactions on Knowledge and Data Engineering*, **5**(6), 990–992.

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*, **3**(2), 105–129.

GAINES, B. R. & MALCOLM, N. (1993). Supporting collaboration in digital journal production. *Journal of Organizational Computing*, **3**(2), 195–213.

GAINES, B. R. & NORRIE, D. H. (1994). Mediator: information and knowledge management for the virtual factory. *SIGMAN AAAI-94 Workshop: Reasoning about the Shop Floor*, pp. 30–39. Menlo Park, CA: AAAI.

GAINES, B. R., RAPPAPORT, A. & SHAW, M. L. G. (1992). Combining paradigms in knowledge engineering. *Data and Knowledge Engineering*, **9**, 1–18.

GAINES, B. R. & SHAW, M. L. G. (1980). New directions in the analysis and interactive elicitation of personal construct systems. *International Journal of Man–Machine Studies*, **13**, 81–116.

GAINES, B. R. & SHAW, M. L. G. (1992). Documents as expert systems. In M. A. BRAMER & R. W. MILNE, Eds. *Research and Development in Expert Systems IX. Proceedings of*

British Computer Society Expert Systems Conference, pp. 331–349. Cambridge: Cambridge University Press.

GAINES, B. R. & SHAW, M. L. G. (1993*a*). Basing knowledge acquisition tools in personal construct psychology. *Knowledge Engineering Review*, **8**(1), 49–85.

GAINES, B. R. & SHAW, M. L. G. (1993*b*). Eliciting knowledge and transferring it effectively to knowledge-based systems. *IEEE Transactions on Knowledge and Data Engineering*, **5**(1), 4–14.

GAINES, B. R. & SHAW, M. L. G. (1993*c*). Open architecture multimedia documents. *Proceedings of ACM Multimedia'93*, pp. 137–146.

GAINES, B. R. & SHAW, M. L. G. (1993*d*). Supporting the creativity cycle through visual languages. *AAAI Spring Symposium: AI and Creativity*, pp. 155–162. Menlo Park, CA: AAAI.

GAINES, B. R. & SHAW, M. L. G. (1994). Using knowledge acquisition and representation tools to support scientific communities. *AAAI'94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 707–714. Menlo Park, CA: AAAI Press/MIT Press.

GIORDANO, L. & MARTELLI, A. (1994). On cumulative default logics. *Artificial Intelligence*, **66**(1), 161–179.

GOODMAN, D. (1990). *The Complete HyperCard Handbook*. Toronto: Bantam.

GOODMAN, D. (1993). *The Complete AppleScript Handbook*. New York: Random House.

GRÄTZER, G. (1971). *Lattice Theory*. San Francisco: Freeman.

GREIF, I., Ed. (1988). *Computer-Supported Cooperative Work: A Book of Readings*. San Mateo, CA: Morgan Kaufmann.

GUPTA, R. (1991). *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD*. Englewood Cliffs, NJ: Prentice-Hall.

HALPERN, J. Y. & VARDI, M. Y. (1991). Model checking vs. theorem proving: a manifesto. In V. LIFSCHITZ, Ed. *Artificial Intelligence and the Mathematical Theory of Computation*, pp. 151–176. New York: Academic Press.

HENKIN, L., MONK, J. D. & TARSKI, A. (1971). *Cylindric Algebras*. Amsterdam: North-Holland.

HILTZ, S. R. (1984). *Online Communities: A Case Study of the Office of the Future*. Norwood, NJ: Ablex.

HOBBS, J. R. & MOORE, R. C., Ed. (1985). *Formal Theories of the Commonsense World*. Norwood, NJ: Ablex.

HUMPHREY, W. S. (1989). *Managing the Software Process*. Reading, MA: Addison-Wesley.

HUNT, C. (1992). *TCP/IP Network Administration*. Sebastopol, CA: O'Reilly.

IMIELINSKI, T. & LIPSKI, W. (1984). The relational model of data and cylindric algebras. *Journal of Computer System Sciences*, **28**(1), 80–102.

KELLY, G. A. (1955). *The Psychology of Personal Constructs*. New York: Norton.

KOLODNER, J., Ed. (1988). *Proceedings Case-Based Reasoning Workshop*. San Mateo, CA: Morgan Kaufmann.

KROL, E. (1992). *The Whole Internet User's Guide and Catalog*. Sebastopol, CA: O'Reilly.

KUMAR, D., Ed. (1990). *Current Trends in SNePS—Semantic Network Processing System*. Berlin, Springer.

LEHRBERGER, J. & BOURBEAU, L. (1988). *Machine Translation: Linguistic Characteristics of MT Systems and General Methodology of Evaluation*. Amsterdam: John Benjamins.

LEVESQUE, H. J. & BRACHMAN, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, **3**, 78–93.

LINSTER, M., Ed. (1991). *Sisyphus Working Papers Part 2: Models of Problem Solving*. Bonn: GMD.

LORBER, J., Ed. (1994). *Paradoxes of Gender*. New Haven, CT: Yale University Press.

LUSCHEI, E. C. (1962). *The Logical Systems of Lesniewski*. Amsterdam: North-Holland.

MACGREGOR, R. M. (1991). The evolving technology of classification-based knowledge representation systems. In J. F. SOWA, Ed. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pp. 385–400. San Mateo, CA: Morgan Kaufmann.

MAIDA, A. S. & SHAPIRO, S. C. (1982). Intensional concepts in propositional semantic networks. *Cognitive Science*, **6**(4), 291–330.

MALCOLM, N. (1991). *GroupWriter: A Word Processor for Collaborative Document Preparation*. MSc. Thesis, University of Calgary, Calgary, Canada.

MICHALSKI, R. S., CARBONELL, J. G. & MITCHELL, T. M., Eds (1986). *Machine Learning: An Artificial Intelligence Approach Volume II*. Los Altos, CA: Morgan Kaufmann.

MICROSOFT (1994). *OLE 2 Programmer's Reference Volume 1: Working with Windows Objects*. Redmond, WA: Microsoft Press.

MINKER, J., Ed. (1988). *Foundations of Deductive Databases and Logic Programming*. San Mateo, CA: Morgan Kaufmann.

MORIK, K., Ed. (1989). *Knowledge Representation and Organization in Machine Learning*. Berlin: Springer.

NEBEL, B. (1988). Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, **34**, 371–383.

NEBEL, B. (1990). *Reasoning and Revision in Hybrid Representation Systems*. Berlin: Springer.

NEBEL, B. & SMOLKA, G. (1990). Representation and reasoning with attributive descriptions. In K. H. BLÄSIUS, U. HEDSTÜCK & C.-R ROLLINGER, Eds. *Sorts and Types in Artificial Intelligence*, pp. 112–139. Berlin: Springer.

ODDY, R. N., ROBERTSON, S. E., RIJSBERGEN, C. J. VAN & WILLIAMS, P. W., Eds (1981). *Information Retrieval Research*. London: Butterworths.

OUSTERHOUT, J. K. (1994). *Tcl and the Tk Toolkit*. Reading, MA: Addison-Wesley.

OXTOBY, J. C., Ed. (1971). *Measure and Category*. New York: Springer.

PATEL-SCHNEIDER, P. F., MCGUINESS, D. L., BRACHMAN, R. J., RESNICK, L. A. & BORGIDA, A. (1991). The CLASSIC knowledge representation system: guiding principles and implementation rationale. *ACM SIGART Bulletin*, **2**(3), 108–109.

PINSON, L. J. & WIENER, R. S., Eds (1990). *Applications of Object-Oriented Programming*. Reading, MA: Addison-Wesley.

QUANTZ, J. J. & ROYER, V. (1992). A preference semantics for defaults in terminological logics. *Proceedings of KR'92: Third International Conference on Principles of Knowledge Representation and Reasoning*, pp. 294–305. San Mateo, CA: Morgan Kaufmann.

QUARTERMAN, J. S. (1990). *The Matrix: Computer Networks and Conferencing Systems Worldwide*. Reading, MA: Digital Press.

QUILLIAN, M. R. (1968). Semantic memory. In M. MINSKY, Ed. *Semantic Information Processing*, pp. 216–270. Cambridge, MA: MIT Press.

REED, A. (1984). Anatomy of a text analysis package. *Computer Languages*, **9**(2), 89–96.

REITER, R. (1980). A logic for default reasoning. *Artificial Intelligence*, **13**, 81–132.

RESNICK, L. A., BORGIDA, A., BRACHMAN, R. J., MCGUINESS, D. L. & PATEL-SCHNEIDER, P. F. (1990). *CLASSIC Description and Reference Manual for the COMMON LISP Implementation Version 1.02*. Murray Hill, NJ: AT&T Bell Laboratories.

RESNICK, L. A., BORGIDA, A., BRACHMAN, R. J., MCGUINESS, D. L., PATEL-SCHNEIDER, P. F. & ZALONDEK, K. C. (1993). *CLASSIC Description and Reference Manual for the COMMON LISP Implementation Version 2.1*. Murray Hill, NJ: AT&T Bell Laboratories.

SCHMIDT-SCHAUß, M. (1989). Subsumption in KL-ONE is undecidable. *Proceedings of KR'89: First International Conference on Principles of Knowledge Representation and Reasoning*, pp. 421–431. San Mateo, CA: Morgan Kaufmann.

SCHMOLZE, J. G. & WOODS, W. A. (1992). The KL-ONE family. In F. LEHMANN, Ed. *Semantic Networks in Artificial Intelligence*, pp. 133–177. Oxford: Pergamon Press.

SCOTT, D. (1976). Data types as lattices. *SIAM Journal Computing*, **5**(3), 522–587.

SHAW, M. L. G. (1979). Conversational heuristics for enhancing personal understanding of the world. In B. R. GAINES, Ed. *General Systems Research: A Science, A Methodology, A Technology*, pp. 270–277. Louisville, KY: Society for General Systems Research.

SHAW, M. L. G. (1980). *On Becoming A Personal Scientist: Interactive Computer Elicitation of Personal Models Of The World*. London: Academic Press.

SHAW, M. L. G., Ed. (1981). *Recent Advances in Personal Construct Technology*. London: Academic Press.

SHAW, M. L. G. & GAINES, B. R. (1983). A computer aid to knowledge engineering. *Proceedings of British Computer Society Conference on Expert Systems*, pp. 263–271. Cambridge: British Computer Society.

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.

SHAW, M. L. G. & GAINES, B. R. (1993). Group knowledge elicitation over networks. In M. A. BRAMER & A. L. MACINTOSH, Eds. *Research and Development in Expert Systems X. Proceedings of British Computer Society Expert Systems Conference*, pp. 43–62. Cambridge: Cambridge University Press.

SHU, N. C., Ed. (1988). *Visual Programming*. New York: Van Nostrand Reinhold.

SIMONS, P. (1982). A Brentanian basis for Lesniewskian logic. In P. GOCHET, Ed. *Logique et Analyse*. Brussels: Centre Nationale Belge de Recherche de Logique.

SIMONS, P. (1987). Brentano's reform of logic. *Topoi*, **6**, 25–38.

SMOLKA, G. (1992). Feature-constraint logics for unification grammars. *Journal of Logic Programming*, **12**(1-2), 51–87.

SOWA, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA: Addison-Wesley.

SOWA, J. F., Ed. (1991). *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. San Mateo, CA: Morgan Kaufmann.

STROUSTRUP, B. (1991). *The C++ Programming Language*, 2nd Ed. Reading, MA: Addison-Wesley.

TARSKI, A. & GIVANT, S. (1987). *A Formalization of Set Theory without Variables*. Providence, RI: American Mathematical Society.

THIMBLEBY, H. (1990). *User Interface Design*. Wokingham: Addison-Wesley.

TRICKEY, H. (1988). C++ versus Lisp: a case study. *Usenix Proceeding C++ Workshop*, pp. 440–449. Berkeley, CA: Usenix Association.

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, DC: IEEE Computer Society Press.

TSANG, E. (1993). *Foundations of Constraint Satisfaction*. London: Academic Press.

VERVEST, P. H. M. (1988). *Innovations in Electronic Mail*. New York: North Holland.

VICKERS, J. N. (1990). *Instructional Design for Teaching Physical Activities: A Knowledge Structures Approach*. Champaign, IL: Human Kinetics.

VICKERS, J. N. & GAINES, B. R. (1988). A comparison of books and hypermedia for knowledge-based sports coaching. *Microcomputers for Information Management*, **5**(1), 29–44.

VILAIN, M. (1991). Deduction as parsing: tractable classification in the KL-ONE framework. *AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 464–470. Menlo Park, CA: AAAI Press/MIT Press.

VLIET, J. C., Ed. (1988). *Document Manipulation and Typography*. Cambridge: Cambridge University Press.

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*. Bonn: GMD.

WALDO, J., Ed. (1993). *The evolution of C++*. Cambridge, MA: MIT Press.

WENGER, E. (1987). *Artificial Intelligence and Tutoring Systems*. Los Altos, CA: Morgan Kaufmann.

WHISSEL, C. M. (1989). The dictionary of affect in language. In R. PLUTCHIK & H. KELLERMAN, Eds. *Emotion Theory, Research and Experience*, Volume 4, pp. 113–131. San Diego, CA: Academic Press.

WINER, D. (1992). *UserLand Frontier User Guide*. Palo Alto, CA: UserLand Software.

WISSKIRCHEN, P. (1990). *Object-Oriented Graphics*. Berlin: Springer.

WÓJCICKI, R. (1988). *Theory of Logical Calculi*. Dordrecht: Kluwer.

WOODS, W. A. (1975). What's in a link: foundations for semantic networks. In D. G. BOBROW & A. M. COLLINS, Eds. *Representation and Understanding: Studies in Cognitive Science*, pp. 35–82. New York: Academic Press.

WRIGHT, J. R., WEIXELBAUM, E. S., VESONDER, G. T., BROWN, K. E., PALMER, S. R., BERMAN, J. I. & MOORE, H. H. (1993). A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems. *AI Magazine*, **14**(3), 69–80.

ZALTA, E.N. (1988). *Intensional Logic and the Metaphysics of Intentionality*. Cambridge, MA: MIT Press.