Tools for Representing and Managing Knowledge: Some Practical Requirements and Suggestions

Larry H. Reeker
Knowledge Models and Cognitive Systems
National Science Foundation
Arlington, VA 22230
lreeker@nsf.gov

Abstract

This is not a technical paper, but rather a suggestion of practical needs in the knowledge representation field. It first attempts to put knowledge representation in a broader perspective, arguing for the importance of software engineering and programming language considerations and advocating inclusion of a variety of representation schemes used in related informational areas. It suggests that only the development of appropriate tools will enable the satisfactory use and management of the needed representations.

Introduction

At the National Science Foundation, the “core” program of research support for artificial intelligence is called “Knowledge Models and Cognitive Systems”. Within this program, Knowledge Representation is recognized as one of three basic research areas. Work in knowledge representation is also fundamental in the other two areas, Cognitive Processing Systems and Knowledge Acquisition and Machine Learning.

Knowledge representation of one type or another has always been a major part of AI, though it was not always recognized how fundamental it was, as a search for papers with the term in the title will quickly reveal. In the early days, the knowledge was just considered part of a computer program, embedded in the code specifying processes or data structures. In the late 1960’s, “problem representation” was isolated for study because of the desire to create general programs that could deal uniformly with a variety of problems. A memorable seminar at Carnegie Mellon University (before it was called that) involved Allen Newell, Herbert Simon, Saul Amarel (who was visiting for the semester) and others in studying the topic. The idea that a problem’s representation could be crucial to its solution was, of course, common to other applied computing areas and was also a necessary concern in the programming language community.

At about the same time, notions of finding representations for natural language utterances were being developed in the linguistics and psycholinguistics communities, and cognitive psychologists were interested in models of how concepts might be related, an example being Quillian’s semantic nets.

With the success of specialized knowledge based systems, knowledge representation came into its own. It was increasingly recognized as an important key to natural language processing, where meaning representation of a structural nature was an aspect of the Generative Semantics school of linguistics. Schank’s Conceptual Dependency was an independently-developed approach to meaning representation which was later generalized to wider knowledge representation tasks.

Figure 1 tries to give a feeling for the flavor of early developments in knowledge representation, without attempting to be complete. It includes contributions from the database community like Entity-Relationship Models and general process description techniques like functional decomposition, which was recognized in the general programming language and software engineering communities but also exploited in the AI community because of the dominance of Lisp.

One of the more successful contributions of the programming language community stems from the late 1960’s but was only recognized by a small community until the mid 1970’s. That was the simulation language Simula, which introduced object-oriented techniques. More will be said of these below.

In the last decade, there have been too many interesting papers on knowledge representation to list individually, though I suggest [1] and [2], and one [9] is singled out in Figure 1 for mention because I want to make a point with respect to it. (There were actually several versions of that paper, one of which was quite a bit earlier, but this is the most accessible.)
Practical programming considerations in knowledge representation

In his 1986 paper [9], Woods considered two major aspects of knowledge representation, expressive adequacy and notational efficacy (see Figure 2). Expressive adequacy is the one that we tend to think of first: Is the notation sufficient to express the knowledge distinctions that we need for our purposes? He pointed out that we also need to consider what distinctions can be left unspecified. But it is on notational efficacy that we want to concentrate.

The first component of notational efficacy, for Wood, is computational efficiency, which is the primary consideration in, say, a problem representation. Does the representation lend itself to the sorts of computations we want to do using the knowledge represented? But it is the other division that is less obvious and often neglected. This is what he calls conceptual efficiency. As Figure 2 shows, it is subdivided into conceptual clarity, conciseness of representations, and ease of modification. Woods discusses these aspects (briefly) in terms of the operation of whatever system is going to be using them, but these same terms also describe important considerations in the human’s ability to understand the knowledge representation. And the ability of a human to understand a computational representation, be it a program or a knowledge base, is the prime consideration in software engineering.

Knowledge representation outside artificial intelligence

There is a lot of representation of knowledge outside of AI. One major growth area of the last decade has been the modeling of information and of information flows in organizations for purposes of strategic data planning. This involves business function modeling if the flows are to be adequately attributed to functions, as well as particular organizations, and can be very complex, since the functions may or may not correspond precisely to organizational entities. There is a lot of knowledge to be captured in this way that has much value to businesses and governmental organizations. In addition to the data planning uses, it is of value in business process engineering or re-engineering and as an archival resource. We can expect this form of knowledge gathering to increase.

---

**Figure 1. An oversimplified history of knowledge representation**

- 1960's - Program Knowledge:
  - Processes
  - Data Structures
  - Linguistic Semantics
  - Ad Hoc Knowledge Bases

- 1970's - Linguistic Meaning Representation
  - Production Rules (Processes)
  - Denotational, Procedural Semantics
  - KL-ONE
  - Entity-Relationship Models
  - Functional Decomposition

- 1980’s - Knowledge Representations
  - Object-Oriented Models

- 1990's - Standardization Efforts
  - Protocols
  - Representation Translation

![Figure 2. Notational Desiderata (Woods, 86)](image)
Within the computer-integrated manufacturing community, there is also a good deal of knowledge gathering. In this case, it is design knowledge, manufacturing knowledge, materials knowledge, etc. This will also increase.

With the advent of enormous on-line digital libraries, the need to index a great variety of material as to content will be greater than ever. Many people assume that some sort of context-based retrieval will suffice, but for any context-based scheme with reasonable recall, the poor precision will swamp the user. It is to be expected that complex items will be described by knowledge representations used to tag the items in such a way that retrieval with reasonable precision can be achieved.

There are other areas, of course, that will increasingly be encoding knowledge, and these areas will tend to develop their own languages unless others are made known to them and are adequately habitable. A family of languages of some generality, under the collective name IDEF, has been developed under sponsorship of the U. S. Air Force, largely by the Knowledge Based Systems Laboratory at Texas A&M University. The original IDEF languages were intended to be useful in integrated computer-aided manufacturing, but it soon became clear they had much wider applications. IDEF0 grew out of the functional decomposition software design paradigm and is used to describe processes and flows of data and control among processes. IDEF1 was designed to describe organizational information flows, and IDEF1x is basically an entity-relationship data modeling language. IDEF0, IDEF1, and IDEF1x are described in various books. IDEF2 is not used. IDEF3 [4] and IDEF4 [6] are less well-known. IDEF3 is designed to capture temporal sequencing and causality. IDEF4 is an object-oriented language of considerable generality that combines elements of the others. The IDEF languages have been used quite successfully to develop large descriptions of systems, mostly enterprise models. The descriptive powers of the family make it likely that they will be transitioned to other areas of knowledge representation, though they may, in their present form, be found lacking by the AI knowledge representation community.

The IDEF languages are visually oriented, rather than textually oriented. This accounts for a lot of their naturalness. Because they are visually oriented, any major use of them must be mediated by the computer. They provide an example of the way that knowledge representation must go. In my view, the visual representation should be primary, as it is in IDEF languages. In many AI systems, the underlying representation is linear, textual. It may be Prolog or some other logical form. I do not believe that textual approaches will stand the test of time.

Suggestions from programming languages

In an earlier paper [7], I suggested that the notions of programming languages and knowledge representations were going to merge to a large degree, since for either to be truly general, it needed to be able to express concepts of almost arbitrary complexity, and the criteria for representing these concepts were always basically the same - what Woods would call expressive adequacy and notational efficacy. Though the importance of the "software engineering" criteria had been recognized less in the knowledge representation area than in programming languages, this was because really large systems that had to be maintained by people who were not the designers were less common in AI than in other application programs. I maintained that "...Knowledge representation is, on its face, a broader term, since knowledge can be that of activities or processes, of which programming is a species, so KR subsumes PL; but knowledge representation is programming, and the success of knowledge systems in practice requires that the incredible potential complexity of bodies of knowledge represented in computers be managed in ways that have been discovered through sad experiences" in other programming domains. I repeat this because it is an important point, an important reason to radically change our ideas of how to handle representation.

The point of the need for software engineering considerations to be embodied in the notation, as well as in the compositional style and in the available tools for representation cannot be overstated. Many knowledge systems that people dream of developing and deploying (and today, these are not just AI experts, since developments in the field have fueled the popular imagination), will be an order of magnitude more complex than typical deployed software today. They may not appear that way if we regard only the system software, which seems a natural extension of numerical computing, database technology, deductive inference programs, known graphical techniques, etc.; but the demands of making available the amount of technical information at the level of detail required will dwarf those of, say, today's management information systems. Experience in the practical deployment of knowledge-based systems is lacking, and virtually nothing is known about the life-cycle knowledge management difficulties and costs of such systems. The new dimension of knowledge base maintenance problems must be added to the ordinary
problems of maintaining large software systems.

Some of the specifically important approaches that I cited from the PL field were (1) Object-oriented approaches, (2) Pictorial programming, (3) Integrating language and programming environment, and (4) availability of distributed and parallel constructs. Clearly, some of these approaches are not well-developed or widely-used in general application programming practice, though they are generally accepted to be worthwhile by people who have studied them, and I expect that they will all become common. As an example, object-oriented programming, which has almost become a cliche, is not as widely practiced as a casual observer might think.

I am convinced that a pictorial approach, with appropriate tools, can provide a superior environment for programming at both specification and implementation levels, and could even enable a smooth machine-aided transition between the levels. The design of the tools available to enable this is not simple, but then, neither is the design of a really good point-and-click interface to any software system, as users of networks and other software will attest, and the general programming environment is a more complex problem. In one exercise several years ago, an approach to visual programming that I had developed [6] was used to develop a knowledge acquisition system [8], and the tool, while far from perfect, is still in use after various modifications.

An important consideration in thinking about managing the complexity of knowledge representations or any complex system descriptions, is the fact that suitably designed interactive systems can be an important aid. When we work with textual code today and manipulate it using text editors, we are working in an unnecessarily impoverished environment.

Tools for knowledge representation

My bias should be by now be clear toward representing knowledge in visual ways, using a computational environment as a mediator. Actually, there are a number of computational tools that would be useful, perhaps essential, in the knowledge representation world of the future. Let us consider a few.

There will need to be discovery tools for knowledge acquisition. This is an area where there has already been considerable work. It should be possible to directly produce the knowledge representation in the tool, and this is generally the case with present tools.

For the present, there will need to be translation tools from one knowledge representation to another if knowledge is to be exchanged. In any case, there needs to be translation for legacy databases or knowledge bases that will be incorporated into new systems, just as a practical matter. Even as some knowledge representation protocols are developed, there will be stylistic differences, as there are in language. It may be desirable to be able to translate locally from nominal (object-oriented) to verbal (functional) style, or vice versa, as a means of better understanding the knowledge base. (This suggestion is based on my belief that neither totally object-oriented or totally functional expression is necessarily the best for every situation, but that is a subject I will not pursue here.)

There will be a variety of diagnosis tools for debugging knowledge bases. We know that such tools can be useful in finding program bugs. Some day, there will also be tools for verifying properties of knowledge bases.

Composition tools will allow the merging of knowledge bases. No matter what the notation, it is too much to expect that separately-composed knowledge bases will merge gracefully.

And there will be viewing tools. In fact, maybe all of the tools will employ viewing, if we adopt non-textual knowledge representations, as I hope we will.

A little thought should make it clear that the tools we are talking about should be as intelligent as possible themselves. They should also have much thought put into the human interface.

Conclusion

The efforts now being undertaken to provide some commonality within the knowledge representations used by the AI community are important [5]. If, at the very least, they provide a vehicle with the utility of ALGOL-60, which served as a means of sharing algorithms even among those who did not use it in writing their own programs and later spawned such important "next steps" in programming languages as Pascal and Simula, they will enrich and advance the field immensely.

As knowledge representation commonality efforts go forward, we must keep in mind the important issues of software engineering that we sometimes overlook in our aesthetic delight with elegant notations. We need to consider the large-scale knowledge bases of the future. We also need to be able to deal with knowledge models which may not be called that because they are used primarily in
other fields. These may use, for instance, design, manufacturing, data planning, and business process modeling notations. These models will themselves benefit from the sophistication of the AI community and AI will benefit from considering notations that are found effective in the other domains.

Clearly, the requirements for successful knowledge representation efforts constitute a tall order. I suspect that to fill that order well, we have to reexamine some of our notions of knowledge representation and of programming languages. Just as the common computer interface for most people today is visual, rather than textual, I would say that knowledge representations should be too. This means handling them through machines. We cannot do things with a point and click interface or video game without interacting directly with the computer. Dealing with knowledge representation will someday be much the same. So will all other types of programming. Truly excellent interfaces are a challenge that can only be answered with both machine intelligence and careful study of human needs and abilities. The same is true of tools for handling knowledge representation.

There are plenty of challenges in knowledge representation, and facing these challenges is more important than the general public or the press realizes. A better understanding of codifying and sharing knowledge and a set of tools to put these to use over very large amounts of information will turn out to be a key that we must have as a preliminary to pleasurable travel on the much-heralded information superhighway of the future.

References


