Luger, G.F. (1994). Forward. In *Expert Systems: Design and Development,* John Durkin. New York: Macmillan.

Foreword

Artificial Intelligence may be defined as: The study of the mechanisms underlying intelligent behavior through the construction and evaluation of artifacts that enact those mechanisms.

Artificial Intelligence is both a science and an engineering discipline. It is a science in that it seeks explication of the nature of knowledge, understanding, and skill. AI also offers a paradigm for testing our understanding of and interaction with the world. We design experiments and run them. The results of our experiments demonstrate our understanding of the world; our revised experiments show our improved comprehension. Artificial Intelligence is engineering in that its practice requires appropriate, flexible, and timely solutions: the design and building of programs and applications that work.

AI has now successfully delivered to the user community many successful design and programming tools; these include algorithms for machine learning, planning and robotics, including the building of closed loop control systems for manufacturing processes. Above all else the Artificial Intelligence research community has developed and delivered expert system or "knowledge-based" programming techniques and applications.

Expert systems encode a human expert's knowledge for a computer in such a fashion that this expert program can be run and the knowledge applied where needed. The expert program is built from explicit pieces of knowledge extracted from the human expert. It is modular and can be easily changed when humans discover new approaches to the problem solving or when the needs of the problem solver change. This expert program can explain itself, by describing why some line of questioning is relevant as well as presenting a proof for how it arrived at some conclusion. The program is also heuristic in that it seldom relies on exhaustive search methods but rather considers the data and knowledge of the application much as the human expert does: with confidences, rules of thumb, and encoded experience of the problem application.

John Durkin's book offers one of the best guides available for the design and building of expert systems. John introduces knowledge-based programming techniques as practical, useful, application-oriented tools. His approach presents expert systems as a natural evolution of AI concepts, thus integrating AI theory with the practice and delivery of quality software.

Figure 1 and some text taken with permission from *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, George F. Luger and William A. Stubblefield, Benjamin Cummings, 1993.

Durkin's approach to expert system software is founded on the use of a "conceptual model" to drive his software development. Figure 1 shows how this conceptual model stands between experience in the world and the creation of code for a computer. A good part of Durkin's book describes AI data structures and shows how they are important conceptual tools for problem solving.

To build an expert system we address a domain of knowledge and skill in an application area; this knowledge is often vague or only partially articulated. The knowledge engineer must translate this into a formal language. This process brings with it several important problems:

1. Human skill is practice based. As Aristotle points out in his *Ethics*, "what we have to learn to do, we learn by doing." Skills such as those possessed by medical doctors are learned as much through years of internship and residency, with their constant focus on patients, as they are in anatomy or physiology lectures, where emphasis is on experiment and theory. Delivery of medical care is to a great extent practice driven. And after years of performance, these skills are highly integrated and often not explicitly retrievable.
2. Human expertise often takes the form of knowing *how* to cope in a situation rather than knowing *what* a rational characterization of the situation might be, of developing skilled performance mechanisms rather than a fundamental understanding of what these mechanisms are. An obvious example of this is riding a unicycle: the successful rider is not, in real time, consciously solving multiple sets of simultaneous differential equations to keep in

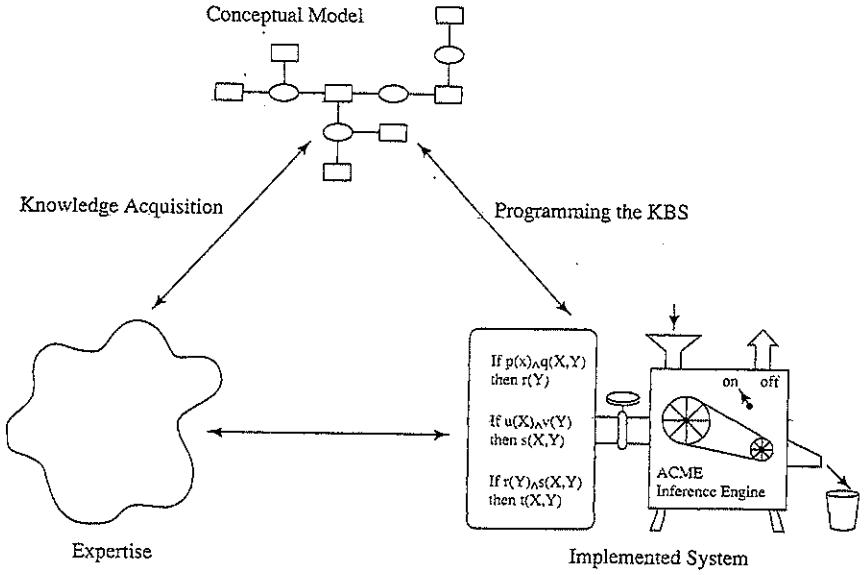

FIGURE 1

balance; rather she is using an intuitive combination of feelings of "gravity," "momentum," and "inertia" to form a usable control procedure. In fact, we find a huge gap often exists between human expertise in an application area and any precise accounting of this skill.

3. We often think of knowledge acquisition as gaining factual knowledge of an objective reality, the so-called "real world." As both theory and practice have shown, there is no immediate access to some "real" world; rather human expertise represents an individual's or community's model of the world. Such models are as influenced by convention, social processes, and hidden agendas as they are by empirical methodologies.
4. Expertise changes. Not only do human experts gain new knowledge, but also existing knowledge may be subject to radical reformulation, as evidenced by ongoing controversies in both scientific and nonscientific fields.

Consequently, knowledge engineering is difficult and should be viewed as spanning the life cycle of any expert system. To simplify this task, it is useful to have, as in Figure 1, a *conceptual* or *mental* model that lies between human expertise and the implemented program. The conceptual model is the knowledge engineer's evolving conception of the domain knowledge. Although this is undoubtedly different from the domain expert's, it is this model that actually underlies the construction of the formal knowledge base.

Because of the complexity and multiple sources of ambiguity in the problem, we should not take this intermediate stage for granted. Expert system builders should document and make public their assumptions about the domain through common software engineering methodologies. A knowledge based system should include a requirements document; however, because of the constraints of exploratory programming, expert system requirements should be treated as co-evolving with the prototype. Data dictionaries, graphic representations of state spaces, and comments in the code itself are all part of this model. By publicizing these design decisions, we reduce errors in both the implementation and the maintenance of the knowledge base.

Knowledge engineers should save recordings of interviews with domain experts. Often, as the knowledge engineer's understanding of the domain grows, she may form a new interpretation or discover new information in one of these sessions. The recordings, along with documentation of the interpretation given them, play a valuable role in reviewing design decisions and testing prototypes.

Finally, this model serves an intermediate role in the formalization of knowledge. The choice of a representation language exerts a strong influence on a knowledge engineer's model of the domain. The model is usually based on one of the AI representation languages, either the predicate calculus, or frames, objects, or hybrid designs.

The conceptual model is not formal or directly executable on a computer. It is an intermediate design construct, a template to begin to constrain and codify human skill. It can, for instance, if the knowledge engineer uses a predicate calculus model, begin as a number of simple networks representing the expert's

states of reasoning through typical problem-solving situations. Only after further refinement does this network become explicit if . . . then . . . rules.

Questions often worked through in the context of a conceptual model include: Is the problem solving deterministic or search based? Is it data-driven, perhaps with a generate-and-test flavor? Is problem solving goal-driven, based on a small set of hypotheses about situations? Are there stages of reasoning? Is it exact or fuzzy? Is it nonmonotonic, with the need of a truth maintenance system?

The eventual users' needs should also be addressed in the context of the conceptual model: What are their expectations of the eventual program? Where is their level of expertise: novice, intermediate, or expert? What levels of explanation are appropriate? What interface best serves their needs?

To accomplish this concept based development process requires:

- Knowledge acquisition, understanding its stages and techniques,
- Tool selection, understanding the many powerful tools available to expert system designers and builders,
- Building the system with steps and cross checks, often employing the exploratory programming methodology,
- Verification and Validation,
- Documentation, and
- Maintenance.

Durkin's book presents excellent suggestions and guidance for performing these tasks.

George F. Luger
Albuquerque