COMPUTABILITY OF DESIGN

Edited by

Yehuda E. Kalay

School of Architecture and Environmental Design
State University of New York
Buffalo, New York

A WILEY-INTERSCIENCE PUBLICATION
JOHN WILEY & SONS
New York / Chichester / Brisbane / Toronto / Singapore

Published by John Wiley & Sons, Inc.

Library of Congress Cataloging In Publication Data:
Computability of design.

(Principles of computer-aided design)
Papers presented at a symposium held at the State University of New York in Buffalo, on December 6-7, 1986.
Bibliography p.
1. Computer-aided design—Congresses.
2. Engineering design—Data processing—Congresses.
I. Kaly, Yehuda E. II. Series: Kaly, Yehuda E.
Principles of computer-aided design.
TA174.C575 1987 620'.00425'0285 87-14745
ISBN 0-471-85387-9

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

9

ARTIFICIAL INTELLIGENCE AND AUTOMATED DESIGN

Stuart C. Shapiro
James Geller

Department of Computer Science
State University of New York at Buffalo
Buffalo, New York

Artificial Intelligence (AI) offers to the design task the use of powerful systems that can be knowledgeable assistants to the human designer. Knowledge Representation techniques can be used to specify the ontology and epistemology of the particular design task so an Intelligent Interface, in general, and an Intelligent Drafting assistant, in particular, can discuss the task with the designer using the same concepts that he uses. Investigating Knowledge Representation formalisms for such aids in the context of developing a Versatile Maintenance Expert System (VMES) has uncovered a number of interesting concepts that seem useful for a wider class of design domains. These concepts are presented after a general discussion of the role of AI in design, and an introduction to a particular AI Knowledge Representation system. The role of design aids and Intelligent Interfaces in VMES is presented as an example of the use of such systems.*

*This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

☐ ARTIFICIAL INTELLIGENCE AND DESIGN

The task of design presents intelligent humans with a large number of complicated problems. Artificial intelligence (AI) is the research area which attempts to discover how to program computers to solve the sort of problems intelligent humans tackle. One use of AI in design might be to have an AI system that would do the design itself, perhaps viewing design as a search through a design-problem space. In this paper, however, we will discuss two aspects of the application of AI as design aids for human designers — the application of Knowledge Representation to drafting systems, and the use of Intelligent Interfaces. After some introductory remarks, we will give a brief introduction to the AI system we are using, present some results of our investigations into the applications of AI to design, and, finally, show how this fits into a Maintenance Expert System we are developing.

The Role of Knowledge Representation

Modern computerized drafting systems supply their users with a wealth of powerful modeling tools. A typical drafting system deals with objects, their visual and non-visual attributes, and their mappings into graphical representations. However, such a system is only a powerful set of pens, it is not an assistant that "knows" what the designer is talking about. To be intelligent, an assistant must be knowledgeable. Knowledgeable computer systems are known as "Knowledge-Based Systems" (KBSs), and are a very active area of AI research and development.

We can identify three roles that people play in the design and use of KBSs. First, there are people who design and implement KBSs without regard to any particular application domain. We can refer to such people as the KBS Designers, and to the results of their efforts, using terminology from the field of Expert Systems (ESs), as "KBS shells." Second, there are those who particularize KBS shells to given application domains. They are called "Knowledge Engineers" (KEs) in the ES world, and we can refer to the results of their efforts as KBSs *simpliciter*. Finally, there are the "end-users" who use KBSs as tools to get particular jobs done.

The job of a KE is usually perceived to be interviewing a person already knowledgeable (at an expert level) in the application domain, and recording that person's knowledge in a form that the KBS shell can use. However, if the KBS shell is flexible enough, there is an additional task for the KE: to design the "form" in which the knowledge is to be recorded. This task is the Knowledge Representation (KR) task, and we will refer to the KE performing this task as the "Knowledge Representation Engineer" (KRE). (The KRE's task has jocularly been called "notational engineering.") The KRE's first task is an analysis of the knowledge primitives in the domain. He must define the domain's ontology (the kinds of objects and attributes contained in the domain), and its epistemology (the sorts of things one may know about the domain, and the ways of knowing them). A flexible KBS shell will permit the KRE to do this declaratively, *i.e.* without reprogramming the shell.

The KRE can supply a vocabulary of conceptual objects, relations, and attributes without a limit on the level of object abstraction. For example, one can take the system's representation of an object, and its representation of the depiction of the object on the screen, and create an explicit non-procedural mapping between them. This mapping itself can be reified, which makes it in turn amenable to serving as an object in a propositional context. This example only involves two levels of abstraction and is frequently useful. For instance, it may be used to assert the validity of a mapping that might be limited to particular circumstances.

The declarative representation of these objects has the additional benefit of placing them in the domain of possible end-user queries. Whatever is a concept for both system and user can be discussed by them. The user can tell the system about them, and can ask the system what it currently knows about them. The system can have rules that specify how to reason about them, how to derive new attributes from old ones, and even under what circumstances to infer the existence of objects it hasn't been explicitly informed about.

A Knowledge-Based drafting system can be an intelligent assistant to a designer, rather than just a powerful drawing tool.

Intelligent Interfaces

Recently, there has been increased interest in the contributions AI can make to the design of interfaces. There was both a workshop and a panel on Intelligent Interfaces at the 1986 AAAI sponsored National Conference on Artificial Intelligence, and DARPA has recently funded a program on Multi-media Interfaces.

Our own view, [Shapiro 1986a] is that an intelligent interface needs the following capabilities: it should know about the topic under discussion, not merely be an isolated, modular, general purpose interface; it should know about communication issues, including what is on the screen, and the relationship between what is being communicated and the way it is being communicated; it should have a user model, so it has an idea of what the user knows, doesn't know, and what the user is trying to accomplish. The KBS-based drafter we are developing can be seen as an appropriate intelligent interface to a more extensive design system.

General Introduction to SNePS

The SNePS Semantic Network Processing System [Shapiro 1979; Shapiro 1986b] is the KBS shell we use, and we will use the SNePS formalism in the

remainder of this paper. For the reader not familiar with SNePS, we will first give a short introduction to the basic properties that distinguish it from other semantic network systems.

SNePS, unlike semantic network systems of the KL-ONE, KRYPTON family, [Brachman 1985; Brachman 1983] but like Anderson and Bower's HAM, [Anderson 1973] is a propositional semantic network system. *i.e.*, the main ingredient of SNePS networks are assertions, constructed from case grammar-like frames [Fillmore 1968]. This does not imply that SNePS cannot support KL-ONE type class hierarchies and inheritance [Tranchell 1982], but that this feature is less prominent in SNePS. SNePS is a fully intensional knowledge representation system [Shapiro 1986b] — it can represent imaginary, non-existing, and even impossible objects, as well as abstract objects, and multiple guises of a single object as if they were separate objects.

SNePS handles full predicate logic with universal, existential, and numeric quantification. A number of non-standard connectives that improve expressibility are available, including both a default operator and a true negation. SNePS supports forward, backward, and bidirectional inference, in contrast to many other systems which permit reasoning in only one direction. For instance, the OPS5 expert system shell does only forward inference, whereas PROLOG does only backward inference. In SNePS, the same rule syntax can be used for either type of reasoning; there are no specific forward or backward rules. SNePS permits the use of recursive rules, either directly recursive or indirectly recursive [McKay 1981]. A relevance logic based [Anderson 1975; Shapiro 1976] extension to SNePS permits its use as a truth maintenance system [Martins 1983].

Another advantage of SNePS is the total order independence of rules and clauses in the rules, in effect eliminating the painful mixed procedural-declarative semantics of PROLOG. This higher degree of flexibility permits very natural representations, especially for natural language rule expressions. However, the required computation times are usually longer than for PROLOG programs.

Although the major purpose of SNePS is not to be a functional model of the brain, as opposed to, for instance, Anderson's ACT system [Anderson 1983], SNePS has been designed with a high degree of cognitive validity in mind. This is expressed by a differentiation between conceptual and non-conceptual relations, by the impossibility of PROLOGish retract-like forced forgetting (except for debugging purposes), and by the accessibility of all information about a concept from the concept itself.

A number of different SNePS interfaces have been designed, containing several natural language parser/generators for subsets of English, a frame-like editor, a logic programming language, and several graphics interfaces. In our description of knowledge structures we will liberally use the "Lispish" notation of the SNePS User Language (SNePSUL), or our standard graphical representation of SNePS networks.

Knowledge Representation in SNePS

In this section, we will discuss an example SNePS network to introduce the syntax and semantics of some of the representational structures we use in our work on VMES, the Versatile Maintenance Expert System [Shapiro 1986c]. Figure 9.1 shows an Adder-Multiplier, a simple experimental device that has been used in the field of hardware maintenance research by a number of people. This object consists of three multipliers and two adders. Figure 9.2 shows part of the semantic network that describes this device. Rectangles in Figure 9.2 represent concepts of real or imaginary objects. Circles represent propositions about these objects. The network can be read as follows: D1 is an object of type M3A2; D1A1 is of type Adder and is a part of D1; D1M1 is of type Multiplier and is a part of D1; D1A1F1 is a Full Adder and is part of D1A1; etc.

The SNePSUL commands that create the network of Figure 9.2 are:

```
(define part-of object type)

(build object  D1A1
       type    Adder
       part-of D1)

(build object  D1M1
       part-of D1
       type    Multiplier)

(build object  D1
       type    M3A2)
       part-of D1A1
       type    Full Adder)

(build object D1A1F1
       part-of D1A1
       type    Full Adder)

(build object  D1A1F2
       part-of D1A1
       type    Full Adder)
```

The first `define` command defines the arcs to be used in the system. Arcs can be followed, for retrieval purposes, in either the forward or backward direction, guaranteeing the universal accessibility of every node from every other node that is related to it.

The set of `build` commands creates the actual network. Note that every `build` command will result in the creation of one "m..." node. These

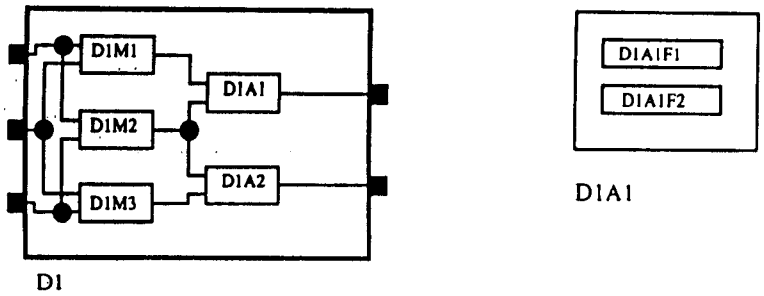

Figure 9.1. The Adder Multiplier and one of its parts.

nodes, as described earlier, correspond to propositions in the system and cannot be created directly by the user. In other words, it is not possible for the user to create an arc connecting two nodes named by him, guarding users against creating non-conceptual propositions, objects the SNePS theory of mind does not permit.

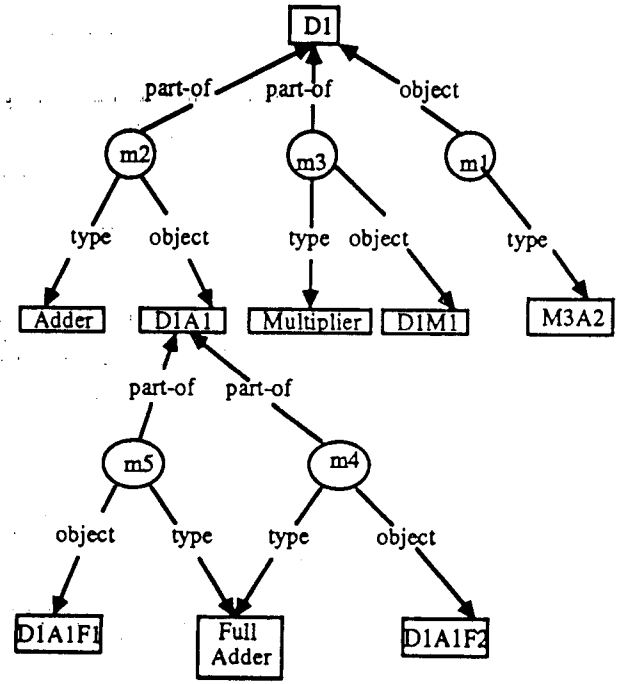

Figure 9.2. A piece of semantic net.

☐ AN ANALYSIS OF IMPORTANT ELEMENTS OF DESIGN KNOWLEDGE

Having introduced the SNePS KBS shell, we will now discuss the ontology and representational constructs that we, in our role as KREs, have found to be necessary for creating descriptions of graphical depictions of simple circuit boards.

Objects and Forms

The first fundamental unit we need to deal with is the displayable *object*. In order to create a picture of an object it is necessary to specify a *form* for it. Every form has a dual role. On the one hand, it can be used to create a picture of that form. On the other hand, it is a conceptual unit in the knowledge representation system and can be manipulated as such. Picture creation is done by a Lisp graphics function whose name is identical to the form concept in the network, and whose arguments are the coordinate positions of the place the form is to be drawn. So, if the form of a particular gate is specified by the function `gate-form`, the gate would be drawn at position (100, 300) by evaluating the Lisp form:

```
(gate-form 100 300)
```

The degree of specificity of a form varies. While the form of an integrated circuit or a transistor is totally fixed, the form of a wire is dependent on the position of the ports it connects. If a user wishes to display an abstract object then he has to supply a symbolic form for it.

Positions

The next essential ingredient for a drafting system is the concept of *positions*. There are several possible ways of specifying positions. In a traditional CAD system, positions are only expressed in an absolute or relative manner based on coordinate values. This is an ability that a KBS should also have. However, knowledge-based design systems should also be able to deal with relational specifications, such as the specification that a certain element should be near or to the left of another element. This, of course, introduces a certain fuzzyness in the representation. However, in many cases this is exactly what a designer would like. It permits him to think in concepts that are natural to him, and it avoids unnecessary specificity. In other words, a knowledge-based drafting system permits one to specify spatial relations with a reasonable degree of imprecision.

The following SNePSUL commands show first our representation for relative coordinate positions, and then for fuzzy positions:

```
(build object   gate-1
       relpos   (build x 100 y 200)
       rel-to   gate-5
       modality function)

(build object   gate-2
       relpos   left
       rel-to   gate-1
       modality function)
```

The first SNePSUL command will create a piece of SNePS network representing the proposition that gate-1 is 100 units to the right and 200 units above gate-5. The second one asserts that gate-1 is to the left of gate-2. The `modality` slot is used to differentiate between different arrangements of an object in a functional representation (wire plan) and a physical representation (picture of the board).

Attributes

Attributes can either be of objects or of pictures of objects. An example of an attribute of a picture is *blinking*. A blinking picture can help a user focus his attention on a currently interesting object, without expressing anything about the object itself.

An example object attribute we have been using is the faultiness of a gate. The proposition that gate-1 is faulty would be represented by the network built by the command:

```
(build object   gate-1
       attr     (build atrb-cls state
                       atrb     faulty
                       modality function))
```

In order that the system know how to display a faulty gate, we tell it that the state attribute maps to the `state-to-color` function:

```
(build attr     state
       mod-func state-to-color)
```

Each attribute function, such as `state-to-color`, is actually a functional that takes a form function and an attribute value as arguments, and returns a modified form function. So, again, if gate-1 had the form represented by the function *gate-form*, and given that gate-1 is in the state of being faulty, and that the attribute function for state is *state-to-color*, gate-1 would be displayed as faulty at coordinate position (100, 300) by evaluating the Lisp form:

```
(funcall (state-to-color #'gate-form 'faulty)
         100 300)
```


Notice that representing different attribute dimensions (state, color, size, etc.) by different attribute functionals explicates the way that different attribute dimensions are, in fact, different.

In this technique, the information of how to display a faulty gate is procedurally encoded in the `state-to-color` functional. An alternative is to store the information declaratively in the network, such as by a proposition built by the command:

```
(build attr     state
       atrb     faulty
       mod-val  red)
```

This proposition says that the attribute of being in a faulty state is to be shown by making the display red. The fact that red is a value of the color attribute is stored by a separate proposition.

Class Hierarchy

An important feature of most knowledge representation systems is their ability to handle *classes* of objects (and also hierarchies with many levels of classes). This permits a user to associate an attribute with an entire class instead of a single object. For example, one could express the fact that all integrated circuits expect ground potential on their pin 0 by associating this fact with the class of all integrated circuits.

Classes have two important features that are valuable for design systems (and KR systems in general). The first is that by asserting that an object belongs to a certain class, a lot of new knowledge is immediately available about it. This is called inheritance along a class hierarchy. The other valuable feature is that this type of representation seems to correspond to the way people organize their knowledge. Therefore the naturalness of the use of classes also improves the general communication between user and system.

Part Hierarchy

Another feature that is common in AI systems is the use of *part hierarchies*. Much of the knowledge about physical objects can be organized as facts that express a part-whole relation between different objects. This applies also and especially to design systems.

Our own research has shown that the concept of inheritability which was mentioned for class hierarchies is also applicable to part hierarchies, but with a difference that we have not seen discussed in the previous literature. For instance, the attribute of a special transistor of being "twice as large as an average transistor" is inheritable by its parts. On the other hand, if a circuit board is known to be faulty, nobody would want this attribute to be inherited by all its parts. That would defeat the very purpose of a diagnosis system.

In class hierarchies, the only attributes that are not inheritable, are those that apply only to classes. For example, the cardinality of a class is not applicable to, let alone inherited by, its individual members. In the part hierarchy, however, there are non-inheritable attributes, such as faultiness, that are applicable to sub-parts, just not to be inherited by all of them.

The representation that we are using for inheritable attributes is the same as the representation for non-inheritable attributes, and is, in fact, identical to the example of faultiness given in an earlier section. However it is possible to assert in the network about a certain attribute that it is inheritable, simply by pointing to it with an `inheritable` arc. For example:

```
(build inheritable size)
```

The display program which interprets the network automatically queries for inheritability if it has to expand an object with attributes into parts. The results of this query determine whether or not the parts of the object are displayed as having the attribute.

Inheritability, as an attribute of other attributes, is a meta-attribute. The fact that we are representing it explicitly and declaratively gives the user the power to experiment with different attributes, and to postpone the decision about which of them is inheritable.

Our findings about inheritance can be extended to other hierarchies, which we refer to as *relevance hierarchies*. Relevance hierarchies are an abstraction of a number of different hierarchies used in the literature, including topic hierarchies [Haan 1986] and hierarchies of spatial universes (containment hierarchies) [Fahlmann 1979].

□ THE VMES SYSTEM

The research described in this paper is a part of the VMES (Versatile Maintenance Expert System) project, which deals with hardware maintenance for mixed analog and digital circuit boards. By using the features of a knowledge based architecture, a high degree of versatility has been achieved [Shapiro 1986c].

The specific significance of our work is that frequently electronic devices have fairly short life cycles. A new board is designed and quickly comes into use in the field. There is little time to design elaborate test procedures or equipment, or to educate a large number of technicians and users. Usually, the only real expert on the device is its designer, and he is already involved with another project when the first problems in the field come up. Our research is directed toward the design of a KBS-based drafter that the designer will use to help design a new device. This design stage will be the "Knowledge Acquisition" stage of the VMES, which will then be able to advise maintenance technicians on the maintenance and repair of the device that it helped design.

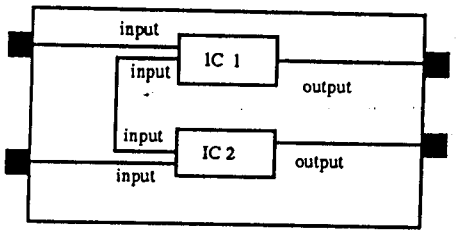

Figure 9.3. An Invalid connection.

The maintenance system can also be used as a part of the design system, since it can be used to detect impossible designs which do not conform to certain integrity constraints. An example of such an impossible design in the circuit board domain would be if a new device that is described to the system has two chips with their input ports connected to each other, but neither connected to an outport of any other chip (Figure 9.3). Another example would be if two points are electrically connected to each other by two separate wires (Figure 9.4).

VMES implements a large number of the concepts which have been described in the previous sections, i.e. part and class hierarchies, inheritance, attributes, etc. It expects to talk to two different types of end-users. On the one hand are maintenance technicians with a limited amount of education and training. On the other hand are the designers that enter a description of a new device into the system. These two types of end-users

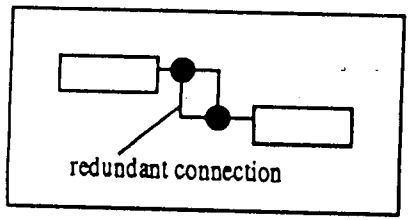

Figure 9.4. A redundant connection.

have different user interfaces, but both interfaces are required to be natural and user-friendly.

The need to create descriptions of circuit boards quickly and without "programming" requires a system that has fairly general knowledge about circuit boards, and that can be adapted to a new device in a short time and with a natural dialogue. To achieve this the system has to *understand* much about the objects of the domain, like wires, inverters or integrated circuits. The use of a Knowledge Representation language is a precondition for achieving such understanding. Use of a component library also permits a rapid change from one device to another. If a new device does not contain any new components, then it is only necessary to describe the new wiring.

Our approach to the design of Intelligent Interfaces may be explained by a description of three interfaces that are part of VMES. The main user interface is a Knowledge-Based graphics component. This program, named `display`, takes a piece of semantic network as argument and uses it to generate a pictorial representation of the stored knowledge. `display` works as a generator, quite comparable with a natural language generator. Only redundant permanent auxiliary storage is used by `display`. In other words, the semantic network plus the Lisp functions describing primitive forms are the only knowledge sources for the computation and creation of device depictions.

We are working on displaying devices under the assumption that no coordinate positions are given. We refer to this activity as intelligent machine drafting (IMD). We are attempting to provide a procedural model of some of the knowledge that a draftsman has about space and arrangement of electronic components. `display` tries to arrange components of the system in what it "thinks" is a graphically appealing way, using several variations of an equal-spacing algorithm. Unlike VLSI routing or layout programs, which usually try to find some space-optimal solution, `display` assumes that there is ample space to solve the placing problem.

The second interface is a natural language understander (NLU), implemented by using an augmented transition network (ATN) [Woods 1970; Shapiro 1982] semantic grammar. A user can create classes of objects, assign (predefined) forms to them, name members of these classes, assign them attributes, and then display them, all with commands from a (fairly limited) subset of natural language. The NLU uses the same KR constructs as are used by `display`. This enables it to demonstrate its understanding of declarative sentences by drawing the object(s) mentioned using appropriate graphic indicators of the asserted attributes.

The third interface is the `readform` facility, which allows a user to create Lisp form functions simply by drawing objects. `readform` permits a user to create pictures of objects from simple primitives like lines, circles, boxes etc. He can also design a form off to the side, on a kind of scratch pad, and then add this form repeatedly to the object being designed. `readform` will assume that the form created on the side is the form of a class of objects, and that the repeatedly added instances are members of that class. These members will also be assumed to be parts of a main object, consisting of the primitives placed before and after using the scratch pad. `readform` verifies some of its assumptions by querying the user, e.g. asking for a name of the suspected class. If the user supplies the requested names then `readform` will create a network structure that asserts the class and part relations and will even store the positions of the parts relative to their super object.

□ CONCLUSIONS

AI offers to the design task the use of powerful Knowledge-Based System shells. Knowledge Representation Engineers particularize these KBS shells to the particular design domain by specifying the ontology and epistemology of the domain. This permits the end-users to discuss the design task with the KBS as if it were a knowledgeable assistant.

We discussed two aspects of KBSs useful for design. Intelligent Interfaces know the task being performed, know about the objects, relations, and attributes being discussed, and know how to express these concepts to the user. Intelligent Machine Drafters (IMDs) are knowledgeable assistants to the designer, besides being powerful drafting tools.

We have been developing a Versatile Maintenance Expert System (VMES) that would be able to help a maintenance technician repair a device that had been designed so recently that there would not have been time to give the technician training on how to repair it. The VMES would acquire its own knowledge of the device by serving as an IMD to the original designer.

In our roles as KREs for VMES, we have identified the following concepts as useful for an IMD and for an Intelligent Interface to a design system: objects; forms of objects; absolute, relative and "fuzzy" positions; attributes of objects and of pictures of objects; attribute functionals; object attribute to picture attribute mappings; class, part, and relevance hierarchies; and meta-attributes, such as inheritability.

□ ACKNOWLEDGMENTS

We would like to thank the other members of the VMES team, Mingruey R. Taie, Sargur N. Srihari, and Scott S. Campbell for valuable discussions; Dale Richards from RADC for administrative support; Bill Eggers, Michael Rosenzweig, Jim Carney, and Carl Mercer for working on several generations of "Readform"; and finally Lynda Spahr, our secretary, for being a pearl in general.

☐ REFERENCES

AAAI *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA, 1986.

Abbott C. "Introduction to the Special Issue on Computer Music," *ACM*, Computing Surveys, 17(2):147-289, June 1985.

Anderson A. and N. Belnap *Entailment: The Logic of Relevance and Necessity*, Princeton University Press, vol.1, 1975.

Anderson J.R. "A Spreading Activation Theory of Memory," *Journal of Verbal Learning and Verbal Behavior*, 22:261-295, 1983.

Anderson J.R. and G.H. Bower *Human Associative Memory*, V. H. Winston and Sons, Washington, D.C., 1973.

Ballard D.H. and C.M. Brown *Computer Vision*, Prentice Hall, 1982.

Brachman R.J., R.E. Fikes and H.J. Levesque "KRYPTON: A Functional Approach to Knowledge Representation," IEEE Computer, 16(10):67-73, 1983.

Brachman R.J. and H.J. Levesque *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Los Altos, CA, 1985.

Boden M.A. *Artificial Intelligence: How Machines Think*, Simon & Schuster, NY, 1985.

Brachman R.J. and J. Schmolze "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, 9(2):171-216, 1985.

Charniak E. and D. McDermott *Introduction to Artificial Intelligence*, Addison Wesley, Reading, MA, 1985.

Fahlmann S.E. *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, MA, 1979.

Fillmore C.J. "The Case for Case," *Universals in Linguistic Theory*, ed. E. Bach and R. T. Harms, Holt, Rinehart, and Winston, NY, pp. 1-88, 1968.

Gardner H. *The Mind's New Science: A History of the Cognitive Revolution*, Basic Books, NY, 1985.

Haan J. de and L.K. Schubert "Inference in a Topically Organized Semantic Net," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, pp. 334-338, 1986.

Hayes-Roth F., D.A. Waterman and D.B. Lenat *Building Expert Systems*, Addison-Wesley, Reading, MA, 1983.

Hofstadter D.R. and D.C. Dennett *The Mind's I*, Bantam Books, NY, 1981.

Hunt M. *The Universe Within*, Simon & Schuster, NY, 1982.

IJCAI *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA, 1985.

McCalla G. and N. Cercone "Approaches to Knowledge Representation," *Computer*, pp. 12-18, Oct. 1983.

McKay D.P. and S.C. Shapiro "Using Active Connection Graphs for Reasoning with Recursive Rules," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, pp. 368-374, 1981.

Martins J.P. *Reasoning in Multiple Belief Systems*, 203, SUNY at Buffalo, Dept. of Comp. Sci., 1983.

Nilson N.J. *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA, 1980.

Peat F.D. *Artificial Intelligence: How Machines Think*, Simon & Schuster, New York, 1985.

Shapiro S.C. and M. Wand *The Relevance of Relevance*, 46, Indiana University, 1976.

Shapiro S.C. "The SNePS Semantic Network Processing System," *Associative Networks: The Representation and use of Knowledge by Computers*, ed. by Nicholas V. Findler, Academic Press, NY, pp. 179-203, 1979.

Shapiro S.C. "Generalized augmented transition network grammars for generation from semantic networks" *The American Journal of Computational Linguistics*, 8(1):12-25, 1982.

Shapiro S.C. and J. Geller "Knowledge Based Interfaces," *AAAI-86 Workshop on Intelligence in Interfaces*, ed. Bob Neches and Tom Kaczmarek pp. 31-36, August, 1986a.

Shapiro S.C., S.N. Srihari, M.R. Taie and J. Geller "VMES: A Network Based Versatile Maintenance Expert System," *Applications of AI to Engineering Problems*, 1986c.

Shapiro S.C. and W.J. Rapaport "SNePS Considered as a Fully Intentional Propositional Semantic Network," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, pp. 278-283, 1986b.

Tranchell L.M. *A SNePS Implementation of KL-ONE*, TR-198, Dept. of Comp. Sci., SUNY at Buffalo, 1982.

Winston P.A. *Artificial Intelligence*, Addison-Wesley, Reading, MA, 1984.

Woods W.A. "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, 10:591-606, 1970.