GRAPHICAL DEEP KNOWLEDGE FOR INTELLIGENT MACHINE DRAFTING

James Geller and Stuart C. Shapiro
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
geller%buffalo@csnet-relay

ABSTRACT*

The problem of Intelligent Machine Drafting is presented, and a description of an existing implementation as part of a graphical generator function is given. The concept of Graphical Deep Knowledge is defined as a representational basis for Intelligent Machine Drafting problems as well as for physical object displays. A (partial) task domain analysis for Graphical Deep Knowledge is presented. Primitives that are necessary to deal with a world of 2-D forms and colors are introduced. Among them are primitives for describing forms, positions, parts, attributes, sub-assemblies, and an abstraction hierarchy. The use of the "linearity principle" for knowledge structure derivation from natural language utterances is shown.

I INTRODUCTION

Traditional computer graphics systems have been criticized in the literature for being a poor environment from a knowledge representation point of view [1]. In graphics as well as in other areas of software development, programmers have been taught not to code any items that are irrelevant to the actual execution of a given task. Over the last several years it has been an essential goal of AI programming to reverse the process of elimination of knowledge from the coding process. The AI programmer tries to make his knowledge conscious and tries to incorporate much of it in his program.

The use of AI techniques in other areas of computer science has led to the replacement of the "eliminate knowledge" paradigm by the "add knowledge paradigm" outside of AI proper. In this sense we interpret Brown et al., and in this sense we want our work to be understood.

In the setting of the VMES project (Versatile Maintenance Expert System) for printed wiring board maintenance [2] we have been working on a knowledge based graphics system. In this paper a new class of layout/routing problems that we have encountered will be described (Intelligent Machine Drafting), and a task domain analysis of what we call Graphical Deep Knowledge will be given.

A. The Display Program

A major part of the VMES user interface is a display program (named TINA), which is called by the maintenance reasoner of VMES and keeps the user constantly informed what VMES is currently "thinking" about. For this purpose it displays, using certain symbol colors, a logical diagram of the circuit board currently being analyzed. Suspected components are displayed in green. Components found faulty are displayed in red. Violated expectations are shown in magenta. Objects about which nothing negative is (yet) known are displayed in blue. Blinking is used to indicate the current focus object of the system.

VMES is implemented on top of SNePS the "Semantic Network Processing System" [3]. The display program is conceived of as a generator that creates pictures from a knowledge base. This is in total analogy to the generation of natural language from a knowledge base.

B. The Linearity Principle of KR

One preliminary idea that has been guiding our work is what we call the *linearity principle*. Let there be two systems S_1 and S_2 consisting both of a parser (P_1, P_2) and a knowledge representation formalism (K_1, K_2). Let there also be a function O_l that can be applied to any natural language expression and a function O_k that can be applied to any knowledge structure of K_1, K_2. O_l and O_k compute some unspecified complexity measure of their arguments.

Both systems S_1 and S_2 impose a mapping from a natural language input to a knowledge structure. We will call these two mappings M_1 and M_2 respectively. We now compute two functions f_1, f_2 such that

$$f_1(u) = \frac{O_k(M_1(u))}{O_l(u)}$$

$$f_2(u) = \frac{O_k(M_2(u))}{O_l(u)}$$

In the above formulas "u" describes a syntactically valid and semantically meaningful natural language utterance contained in the domains of both P_1 and P_2. We will call the system S_1 better than the system S_2 if $f_1(u)$ can be approximated better by a constant than $f_2(u)$. A practical judgement about the constancy of f_1 and f_2 could be done by computing mean and standard deviation of f_1 and f_2 for a large number of different u values, however we will limit ourselves to intuitive judgements.

The Linearity Principle:

> The quotient of the complexity of a knowledge structure and the natural language utterance that it represents should be approximately a constant for any given parser and KR system.

Intuitively the linearity principle says that we do not want to represent most five word sentences of a language with three or four semantic network nodes, but have one five word sentence of this language represented with 25 nodes. An implicit application of the linearity principle (LP) can be seen in Shapiro's work on non-standard connectives [4].

Before we present an example application of the LP it is necessary to say that the arc labels in SNePS networks (Fig. 1, more explanations will be given in the next section) are seen as system primitives. The number of different arc labels is not fixed and can be extended by the user. It is assumed that the number

* 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).

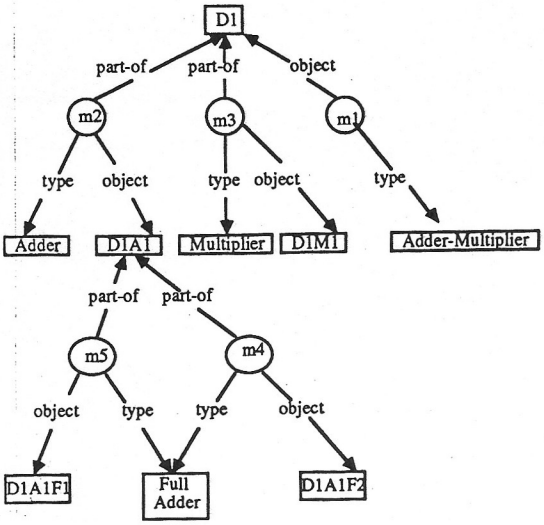


Figure 1

A SNePS network giving (partial) information about an Adder-Multiplier

of arc labels necessary for one limited domain will converge towards a stable set, therefore identifying this set is a way of task domain analysis [5].

If people can describe a simple arrangement of objects by a short sentence then it should be possible to describe it with a reasonably simple SNePS structure. If this is not the case then the number of user defined primitives has to be extended to accommodate the sentence. (Of course new primitives will also have to be used if the sentence is not representable at all).

For instance if two people are sitting in front of a graphics terminal displaying the Adder-Multiplier (which has been used in maintenance research, Fig. 2), and one of them asks:

"Tell me the names of all multipliers."

then the other person will presumably be able to do that. Therefore we would want our graphics interface to be able to do the same thing. We also want the knowledge base to contain information on all multipliers in a format approximately linear in size with respect to the answer given by a person. This leads directly to an old idea, the implementation of a class hierarchy. (Less obvious examples will be given throughout this paper.)

C. Notational Conventions for SNePS networks

Fig. 1 shows an example of a typical SNePS network in order to provide some intuition for the reader not familiar with SNePS. The syntax and semantics of SNePS have been carefully defined [6]. SNePS is also a "neat" KR system that incorporates full first order predicate calculus. We will use Fig. 1 to introduce the network notation that will be used in this paper.

The nodes m1, m2, m3, m4, m5 represent propositions. m2 expresses the fact that the object D1A1 is of type Adder. An equivalent first order predicate calculus representation for Fig. 1 would be the following one:

type(m1,Adder-Multiplier) & object(m1,D1)
type(m2,Adder) & object(m2,D1A1) & part-of(m2,D1)
type(m3,Multiplier) & object(m3,D1M1) & part-of(m3,D1)
type(m4,Full-Adder) & object(m4,D1A1F2) & part-of(m4,D1A1)
type(m5,Full-Adder) & object(m5,D1A1F1) & part-of(m5,D1A1)

This representation is unnecessarily redundant, and we will introduce a pseudo-predicate notation according to the following formal scheme. Given a conjunction of a number of binary predicates with identical first arguments, transform the first argument into a pseudo predicate. Transform all binary predicates into arguments at odd numbered positions, and insert all second arguments at even numbered positions. In symbols:

$$\&_{i=1}^{n} p_i(a_0, a_i) \rightarrow a_0(p_1\ a_1\ p_2\ a_2\ \ldots)$$

This transformation is syntactic sugar and has no influence on the meaning of the representation which depends on the combination of system primitives (arcs). Therefore all the a_i's that will be given in the following sections are to be understood as examples.

II INTELLIGENT MACHINE DRAFTING

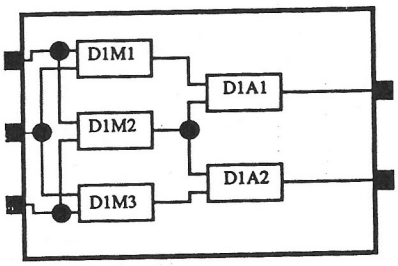
The creation of *logical* circuit board diagrams from a knowledge base is not addressed by a number of commercially available Computer Aided Drafting systems as well as research on CAD, layout systems, and routers [7,8]. Work has concentrated on layout and routing of physical diagrams. Logical diagrams are usually created with a graphics editor or by computer from hand sketches [9].

We are interested in layout and routing of logical circuit diagrams. Physical diagrams created by CAD systems have to be realized in hardware, and therefore the layout is usually optimized for signal length, area consumption, power consumption or heat dissipation. None of these requirements exist for logical diagrams. Rather one wants to create pictures that are optimized in a "human factors" sense [10]. The described difference can best be compared with the shift in attention in programming language research from space and time efficient programs to readable and maintainable languages.

Physical and logical routers also differ in their initial problem setting. A physical router prohibits wire crossings and makes use of different layers and "vias" to avoid them. A logical router permits wire crossings. It uses a special symbol (usually a dark dot at an intersection) to mark clearly whether a crossing is meant to be an electrical connection or not.

Def: Intelligent Machine Drafting (IMD).
Intelligent Machine Drafting is the activity of automatic creation of a cognitively appealing logical diagram of a system from a knowledge base which contains *no numerical coordinates* of the components of the system.

The application of IMD to circuit boards implies the need for a module that creates cognitively appealing layouts of all components and a logical (as opposed to physical) router that connects them.



D1

Figure 2

IMD has turned out to be an interesting problem from a theoretical point of view. It used to be the working method of engineering (and is still common) that a design engineer would send a hand sketch to a draftsman who would then draw a nicely laid out version of it. The job of the draftsman is usually considered a "low intelligence" position, requiring only a minor level of technical education. However this "low intelligence problem" differs from many "hard" AI problems because it cannot be translated easily into a symbolic representation. Solving IMD problems by humans requires use of (part of) the perceptual system, and possibly of the imagery system, both domains which researchers have not yet related well to the domain of problem solving.

A. IMD for circuit board display

We have defined a very limited class of objects called A*M* which is a simple generalization of the Adder-Multiplier of Fig. 2, and we have implemented a fully automatic layout and logical routing program for this class. A formal description of the class A*M* has been formulated. However, we will limit ourselves in this paper to an intuitive explanation.

- A device of class A*M* consists of at most one main object which is assumed to be a large box graphically containing all objects asserted as its parts. In the absence of this main object the screen itself is considered the main object.
- Signal flow in an object of the class A*M* as well as in its parts is strictly from left to right, with no feedback at any stage.
- The "signal length" (maximum number of components a signal has to pass through from system input to system output) and the "signal width" (maximum number of components that are active in parallel, assuming constant delay for every component) are small enough that linear chains of components can be constructed that will fit into the given main object, leaving enough additional space for wiring.
- The main object as well as all its parts have ports. Besides that there is no second level of the part hierarchy. (The ports of the main object are shown as little black boxes in Fig. 2.)

The current implementation makes a few additional assumptions which are not part of the A*M* definition, which however do not impose severe loss of generality and will be eliminated in the future. Among these assumptions are the constancy of the port size for all components and the assumption of small variation of size among components. Forward jumps of connections have not yet been implemented.

For efficiency reasons there is no backtracking programmed into the router, therefore one can design pathologically unsolvable cases rather easily, which does not currently concern us, given that the "general purpose routing problem" has not yet been solved either [7]. Work on IMD as well as on the display of *physical* board diagrams from a knowledge base have motivated our work on graphical deep knowledge (which will be defined in the next sections).

III GRAPHICAL DEEP KNOWLEDGE

A. Definition of Graphical Deep Knowledge

A large number of scientific fields make marginal to extensive statements about the representation of knowledge dealing with forms and colors. Space limits us to mention three main areas, natural language graphics [11], knowledge based graphics [12, 13], and the imagery debate between "imagists" [14, 15] and "propositionalists" [16, 17]. The existence of some propositional representations is now widely accepted in all camps. However, many researchers seem to gloss over the details of their representation, just stating that problem so and so could be done with a list of propositions. Kosslyn's implementation, a notable exception [14], implies an important criterion for a propositional representation of forms, namely that it can be used to create actual pictures. We want to call this criterion *projective adequacy*.

A system that also permits *reasoning* based on shapes or positions of objects will be said to demonstrate *deductive graphical adequacy*. The type of reasoning permitted is either analogical or propositional. In order to avoid any possible terminological confusions we will shun the terms "spatial knowledge", "visual knowledge" and even "graphical knowledge" and use the term "graphical deep knowledge".

Def: Graphical Deep Knowledge
A knowledge base is said to contain graphical deep knowledge if at least part of its knowledge exhibits deductive graphical adequacy, and part of its knowledge exhibits projective adequacy.

The term "deep" is used in analogy with deep structures in linguistics.

One major goal of our research is to create a base of graphical deep knowledge that is adequate for displaying and reasoning about objects in general and about the domain of circuit boards in particular. Our current analysis of graphical deep knowledge is given in the following sections.

B. Form Knowledge

Objects may have individual forms or inherited forms.

m1(object d1 form xand modality function) (1)

m2(object d2 type and-gate modality function) (2)
m3(sub-class and-gate class boolean modality function) (3)
m4(class boolean form xboolean modality function) (4)

(1) describes an object (individual) d1 that has a form xand. The last binary predicate "modality" is used to discriminate between different display modes. Circuit boards permit display of their wire plan (logical or functional representation) and of their physical structure. The forms used for these two displays are usually different, therefore the form proposition must be qualified by the display modality for which this item of knowledge is valid.

A form like "xand" is at the same time a node in the semantic net and a LISP function that, if executed, would draw a specific form. Form functions are parameterized by the starting position. Therefore one form function can display the same object at different positions, but no other modification is possible.

(2) assigns d2 to the class of and-gates which are by (3) recognized as a sub-class of the class of boolean components which by (4) are all assigned the same form, namely xboolean. We have never found it necessary to inherit a form using an intermediate class.

C. Position Specification

A large number of representations for positions is possible. All object positions refer to the position of an object's fixed reference point.

1. Concrete and Fuzzy Absolute Positions

m5(object d1 abspos m6(x 100 y 200) modality function) (5)

(5) describes an absolute position of d1. The position is given by the substructure m6 which contains actual coordinate values.

The pseudo predicate m6 has to be read as a structured individual, not as a proposition [6]. The implicit assumption of this representation is that coordinates are given in pixels of a graphics display device and are "relative to the screen", and therefore called absolute.

When people give a description of a picture, they typically do not use coordinate values but rather talk about objects in the center, at the top, or at the left of the screen. According to the linearity principle it is therefore necessary to represent these "fuzzy" absolute positions. (6) shows an example of a fuzzy absolute position.

```
m7( object d2 fabspos center modality function)          (6)
```

Currently the exact meaning of the fuzzy terms is still under investigation. We have done a psychological pilot study with 20 subjects to find out what people think "leftness" means, which has not yet been totally evaluated. Fuzzy absolute positions used in this experiment are top, bottom, left, right, center, upper left corner, upper right corner, lower left corner, and lower right corner. The term "fuzzy" is not related to Zadeh's fuzzy logic [18].

2. Relative Positions

Propositions about relative positions can be divided into different groups, according to a number of criteria. The first distinction is between numeric positions (what we refer to as "concrete" positions) and fuzzy positions. For numeric positions there are at least three different ways to interpret coordinate values. Values can be given in pixels, or they can be multiples of the sizes of either the object or the reference object involved.

The reference object might be given explicitly or implicitly. In the second case there must be a "super-part" of the object which will be used as the reference object. Finally it might be the case that a relative position is inherited from a class of objects. Many of the given representational possibilities can be combined with each other.

a. Fuzzy Relative Positions--As for fuzzy absolute positions the analysis of the semantics of fuzzy relative positions is still under way and based on experimental data.

```
m8( object d3 frelpos left rel-to d1 modality function)  (7)
```

(7) describes the proposition that d3 is left of d1.

Unfortunately there are a number of fuzzy relative position descriptions which do not rely on binary relations. A representation for "between", which has two reference objects is shown in (8). More difficult are "on-one-line", "together", and "forming-a-circle".

```
m9( object d99                                           (8)
    frelpos between
    rel-to1 d98
    rel-to2 d97
    modality function)
```

b. Concrete Relative Positions--We will begin this section with an example for a relative position measured in pixel coordinates. Fig. 3 shows as example a multiplier. The little black boxes in the picture are ports (sic!) of the multiplier and have their own forms.

```
m10( object port3                                        (9)
     relpos m11( x 24 y 4)
     rel-to D1M1
     modality function)
```

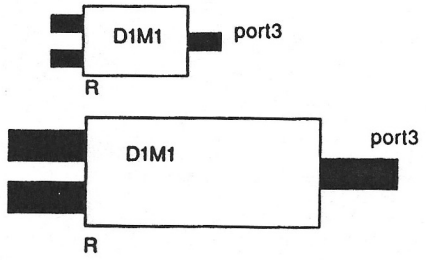


Figure 3
A Multiplier with 3 Ports
in 2 sizes. In both sizes
port3 is one bodylength away from R

(9) describes the relative position of port3 as being 24 to the right of D1M1, and 4 above it. Distances refer to the reference point R. If the relative position of a part of an object is given in pixel coordinates then a problem with scaling results: not only objects have to be scaled, but also relative positions. This is unsatifying because it does not express the fundamental invariance of the position of the sub-part to its super-part. Fortunately it is possible to represent the relation between an object and its sub-parts preserving the conceptual positional invariance by using "body coordinates". These coordinates represent a relative position as multiples of the size of the relevant object.

```
m12( object port3                                        (10)
     relpos m13( bx 3 by 1)
     rel-to D1M1
     modality function)
```

(10) shows the same relative position as (9), however assuming that object port3 has a length of 8 pixels and a width of 4 pixels. The relative position "3", is a multiple of the size of port3. The length and width of an object are the length and width of the smallest surrounding rectangle of it which has lines parallel to the coordinate axes ("extent").

Intuitively, the representation expresses the fact that a big man has his arms far away from his neck, and a small child has its arms near to the neck, but the ratio of the distance and the size of the person should be approximately a constant.

Usually there will be a number of objects given with relative positions to the same reference object. This makes it desirable to specify relative positions in body coordinates of the reference object, shortly called reference object coordinates (denoted by the arcs brx and bry).

```
m14( object port3                                        (11)
     relpos m15( brx 1 bry 0.33)
     rel-to D1M1
     modality function)
```

(11) can be interpreted in the same way as (10), except that this time the factors (after "brx", "bry") apply to the size of the reference object, which is assumed to be 24 pixels long and 12 pixels high.

c. Explicit versus Implicit Reference Objects-- In all cases so far the reference object of a relative position statement was given with a rel-to arc. In the circuit board maintenance domain a flat part hierarchy is used. There is one major object, the board, which has many different parts which should reasonably be placed relative to this main object. It would be redundant to assert the reference object for all the parts, and therefore a default assumption is practical.

m16(object d5 relpos m17(x 100 y–20) modality function) (12)
m18(object d6 sub-parts d5 modality function) (13)

(13) shows a part assertion, a descriptive tool that will be reviewed later on. Because of (13) the relative position asserted by (12) will be interpreted as being relative to d6.

Combinations of the representational constructs introduced are in general possible. For instance an implicit reference object may be used with all types of relative coordinates, including fuzzy ones (14, 15).

m19(object d7 frelpos left modality function) (14)
m20(object d8 sub-parts d7 modality function) (15)

d. Inherited Relative Positions--If one adder has its first port at half a body length from its reference point, then this will presumably hold true for all the adders in the system, and one would not want to assert this over and over again. A solution to this problem is to make the relative position itself inheritable. This option is exemplified by the following set of propositions. The relative position is given in reference object coordinates and inherited through an intermediate class "half-adder".

m21(object d9 type half-adder modality function) (16)
m22(object d10 sub-parts d9 modality function) (17)
m23(sub-class half-adder class adder modality function) (18)
m24(object d10 form xadd modality function) (19)
m25(class adder relpos m26(brx 2 bry .5) modality function) (20)

(16), (18), and (20) specify the relative position of d9 which is inherited from the class "adder"; (17) specifies the reference object d10 by force of its super-part relationship to d9. (19) is necessary to permit the derivation of the size of d10 which in turn is necessary for the computation of reference object coordinates.

3. Logical Reasoning with Fuzzy Positions

The following structure is a SNePS rule that expresses the fact that "if one object is left of another object, then the other object must be right of the first object and vice versa". For a detailed explanation of the structure of SNePS rules, see [19]

m27(avb (v1 v2 v3) (21)
thresh 1
arg (m28 object v1 rel-to v2 modality v3 frelpos left)
arg (m29 object v2 rel-to v1 modality v3 frelpos right))

If the knowledge base contains the absolute position of B and the fuzzy position of B relative to A but no positional information about A itself, then A's fuzzy position can be derived with rule (21) or a variation of it.

D. Parts, Clusters, and Assemblies

Part hierarchies are a commonly used construct in AI [20]. Our research has indicated that a part hierarchy alone is not sufficient for graphical deep knowledge representations. We have added two other types of part-like hierarchies, called assemblies and clusters.

The display of a complicated object with several levels of parts might be impossible on a limited resolution display device. A natural way to limit the complexity of such a display task is to limit the number of levels of the hierarchy that are actually displayed. This is a very elegant solution because it does not require the introduction of any new representational construct.

(17) showed our representation of a simple part relation. An object can of course have more than one part. The ubiquitous modality attains a special importance for part hierarchies. Circuits like AND gates, OR gates etc. are displayed as single objects in a logical diagram. In real hardware there are usually four binary AND gates in a single chip. These four gates might be parts of different logical units. However, in a physical representation all four of them must be parts of the same integrated circuit.

1. Assemblies

Work on the maintenance part of the VMES project has led to the realization that certain objects should never be displayed without their parts. For instance, a port is a part of a multiplier, but a multiplier should never be displayed without its ports. Sub-assemblies are therefore objects that have a real part-whole relation to a specific object and which are supposed to be displayed whenever the object they are part of is displayed.

The representation of sub-assemblies is similar to part-whole relations, except that the arc "sub-assems" is used instead of "sub-parts".

m30(object d10 sub-assems d9 modality function) (22)

2. Clusters

Printed circuit boards sometimes show groups of objects that stand in a logical relation to each other, comparable to a part-whole relation. Nevertheless they are neither sub-parts nor sub-assemblies. Sub-parts and sub-assemblies have a *main object* that is itself displayable, i.e. that has a form. However, a grouping of components might consist of objects of the same size and importance, none of which deserves the status of main object. Fig. 4 shows a voltage divider and a T filter which are typical examples of such circuits.

A grouping which exists only as an abstraction is called a *cluster*. If one combines the concept of cluster with the concept of level a dilemma emerges. Either the abstract object is left out of the hierarchy (which is undesirable, because anything that seems natural to a person should be directly representable in the network (LPI)), or the abstract object is put in the hierarchy and the objects of the cluster are made its parts. But now the idea of creating simplified displays by limiting the number of levels displayed does not work any more, because the abstract object is not displayable in the same sense as real objects are. Moreover if one is willing to give an abstract object a symbolic form, then both the symbolic form as well as the cluster elements would be displayed if one wants to see all the levels of the part hierarchy. This would complicate the display unnecessary.

Our answer to this problem is to create an additional hierarchy which stands somewhere in between a part hierarchy and an abstraction hierarchy. If A is an object (without form) which has sub-clusters B, C, and D then A will be displayed *only* by displaying B, C, and D. However if a partial display is enforced in a way that would exclude the level of B, C, and D from showing, A will be displayed symbolically by a box, akin to the display format in block diagrams. Fig. 5 shows the new display format for Fig. 4. The network representation of a sub-cluster is shown by (23).

m31(object d10 sub-clusters d9 modality function) (23)

E. Attributes and Attribute Mappings

One important factor in designing a system based on graphical deep knowledge is a clear separation between icons and the objects that are represented by these icons, an observation that has been made by others also [21]. This separation forces one to

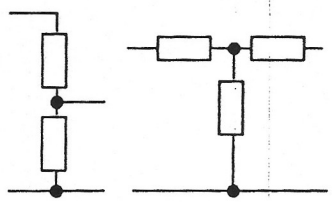


Figure 4

A Voltage Divider A "T"-Filter

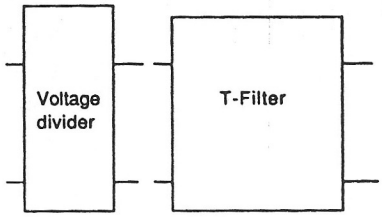


Figure 5

Abstract Representations for Figure 4

distinguish between attributes of objects and attributes of pictures. A typical example of an attribute of a picture is ***blinking***. On the other hand, *faultiness* is an attribute of a component, not of the picture of a component. Attributes like faultiness cannot be displayed directly and therefore have to be symbolized with pictorial attributes.

Attributes are represented by the name of an attribute-class with 0 to 3 positions for attribute values. The following examples show an attribute with no attribute-value (24), an attribute with one position containing the value "faulty" (25), and an attribute with two positions containing the values "left" and "90" (26). No need for any attributes with more than three positions for attribute-values has arisen yet.

```
m32( object d1                                              (24)
     attr m33( atrb-cls new modality function))
m34( object d1                                              (25)
     attr m35( atrb-cls state
               atrb faulty modality function)
m36( object d2                                              (26)
     attr m37( atrb-cls rotated
               atrb1 left atrb2 90 modality function)
```

The attribute statements (24) - (26) apply to objects as opposed to pictures.

Attribute classes are linked to functionals that can be applied to form functions. Such a functional is called a ***modifier function***. (26) asserts that d2 has the attribute-class "rotated" with two values "left" and "90" (degrees), which requires a change to its form before it can be displayed correctly. Therefore a modifier function that rotates forms is bound to this attribute class and the form of d2 is passed to it as first argument. The attribute values (marked by the arcs atrb, atrb1, atrb2 and possibly atrb3) are passed in correct order as additional arguments to the modifier function.

The binding of an attribute-class to a modifier function is asserted in the knowledge base. This makes it amenable to easy change by the user. Utterances like

"Represent the state faulty by red color and the state good by blue color." (27a)

have led to this representational decision (linearity principle!). In (27b) the complete mapping necessary for (27a) is given.

```
m38( attr state                                             (27b)
     mod-func color
     modality function
     val1 m39( expressed faulty expressed-by red)
     val1 m40( expressed good expressed-by blue))
```

It binds the attribute-class "state" to the modifier-function "color". The two sub-structures at the end of the val1 arcs show value mappings between object attributes and picture attributes. The object attribute of faulty state is represented by the picture attribute of red color. val1 corresponds to atrb1 and specifies value mappings that apply to the first attribute position.

Sometimes one encounters numerical attribute values as in (26). Representing a mapping between two large lists of numbers would be unwieldy to impossible. Luckily in many practical cases the relation between the attribute value of a picture and the attribute value of the corresponding object is an identity function. This case is taken care of by using the attribute value of the object if there is no explicit mapping from object value to picture value. This also can take care of a mapping that is the identity function except for a few singularities.

If a more complicated function is necessary to transform from object attribute-value to picture attribute-value then the mapping function has to be integrated in the attribute functional itself.

The method used to associate an actual function with an attribute-class is identical to the method used for form functions. The node specifying a modifier-function is at the same time the name of a LISP function.

An important finding of our work has been that there is inheritance along part hierarchies, something we have not yet seen in the literature. For instance, if an object is represented with an attribute like "scaled", then one would want all its parts to inherit this attribute. Even more interesting is the fact that this inheritance does not apply to all attributes and depends on the attribute-class itself. If one asserts, for instance, the faultiness of an object then it would defeat the whole purpose of a maintenance system to have all its parts inherit faultiness.

Given that one can easily express a fact like "scaling is inheritable", one should also (linearity principle!) have a correspondingly simple representation in the network which is exactly what has been implemented.

```
m41( inheritable size)                                      (28)
```

If (28) is part of the current knowledge base then the attribute-class "size" will apply to the parts of all objects for which the size attribute has been asserted.

IV IMPLEMENTATIONAL STATE

TINA is going through its fourth cycle of implementation which is done in Franz LISP on top of SNePS. All of the shown knowledge structures (and more) are representable and retrievable from the network knowledge base, and most of them are interpreted in a way consistent with the descriptive semantics given in this paper. An older version of "TINA" has been applied to a real circuit board used for telecommunication purposes (PCM board). The IMD system described has been used for the Adder-Multiplier only.

V CONCLUSIONS

The problem of Intelligent Machine Drafting has been introduced, and it was argued that it is a theoretically interesting AI problem which is sufficiently different from other CAD techniques to deserve separate investigation. The class A*M* has been defined informally, and a few additional restrictions of the current IMD implementation for objects of this class have been given. The definition of Graphical Deep Knowledge and a (partial) task domain analysis of this area have been presented. A number of representational primitives have been introduced by way of example. These primitives comprise structures for representing knowledge about forms, concrete and fuzzy positions, and attributes. Positions have been differentiated into absolute and relative positions with explicit and implicit reference objects. Pixel based coordinates, body coordinates and reference-object coordinates have been introduced. Part hierarchies have been discriminated into real part hierarchies, sub-assemblies, and abstraction-hierarchy like clusters of objects. The derivation of some of these structures based on the "linearity principle" has been demonstrated, by presenting examples for motivating natural language utterances. A generator function which creates graphical representations from a knowledge base containing the indicated structures has been implemented.

References

1. D. C. Brown and B. Chandrasekaran, "Design Consideration for Picture Production in a Natural Language Graphics System," *Computer Graphics* **15**(2) pp. 174-207 (July 1981).
2. Stuart C. Shapiro, Sargur N. Srihari, Ming-Ruey Taie, and James Geller, "VMES: A Network Based Versatile Maintenance Expert System," *Proc. of 1st International Conference on Applications of AI to Engineering Problems*, pp. 925-936 Springer Verlag, (April 1986).
3. Stuart C. Shapiro, "The SNePS Semantic Network Processing System," pp. 179-203 in *Associative Networks: The Representation and use of Knowledge by Computers*, ed. Nicholas V. Findler,Academic Press, New York (1979).
4. Stuart C. Shapiro, "Using Non-Standard Connectives and Quantifiers for Representing Deduction Rules in a Semantic Network," *Presented in Tokyo at: Current Aspects of AI Research*, (Aug. 27-28, 1979).
5. Stuart C. Shapiro and James Geller, "Artificial Intelligence and Automated Design," *1986 SUNY Buffalo Symposium on CAD: The Computability of Design*, SUNY at Buffalo, (Dec 6-7, 1986).
6. Stuart C. Shapiro and William J. Rapaport, "SNePS Considered as a Fully Intensional Propositional Semantic Network," *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 278-283 (1986).
7. Hajimu Mori, Fujita Tomoyuki, Masahiro Annaka, Satoshi Goto, and Tatsuo Ohtsuki, "Advanced Interactive Layout Design System for Printed Wiring Boards," pp. 495-523 in *Hardware and Software Concepts in VLSI*, ed. Guy Rabat,Van Nostrand Reinhold, New York (1983).
8. Isao Shirakawa, "Routing High Density Printed Wiring Boards," pp. 452-479 in *Hardware and Software Concepts in VLSI*, ed. Guy Rabat,Van Nostrand Reinhold, New York (1983).
9. Helmut Jansen, Erhard Nullmeier, and Karl-Heinz Roediger, "Handsketching as a Human Factor Aspect in Graphical Interaction," *Computers and Graphics* **9**(3) pp. 195-210 (1985).
10. M. C. Maguire, "A Review of Human Factors Guidelines and Techniques for the Design of Graphical Human-Computer Interfaces," *Computers and Graphics* **9**(3) pp. 221-235 (1985).
11. Michael Hussman and Peter Schefe, "The Design of SWYSS, a Dialgue System for Scene Analysis," pp. 143-201 in *Natural Language Communication with Pictorial Information Systems*, ed. Leonard Bolc, (1984).
12. Frank Zydbel, Noton R. Greenfeld, Martin D. Yonke, and Jeff Gibbons, "An Information Representation System," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 978-984 (1981).
13. Mark Friedell, "Automatic Synthesis of Graphical Object Descriptions," *Computer Graphics* **18**(3) pp. 53-62 (1984).
14. Stephen M. Kosslyn and Steven P. Shwartz, "A Simulation of Visual Imagery," *Cognitive Science* **1** pp. 265-295 (1977).
15. Stephen Michael Kosslyn, *Image and Mind*, Harvard University Press, Cambridge MA (1980).
16. Zenon W. Pylyshyn, "The Imagery Debate: Analogue Media versus Tacit Knowledge," *Psychological Review* **88**(1) pp. 16-45 (1981).
17. Zenon W. Pylyshyn, *Computation and Cognition*, MIT Press, Cambridge MA (1984).
18. Lofti A. Zadeh, "Commonsense Knowledge representation Based on Fuzzy Logic," *Computer*, pp. 61-65 (Oct. 83).
19. Stuart C. Shapiro and The SNePS Implementation Group, "SNePS User's Manual," SNeRG Bibliography #31, SUNY at Buffalo (Sept. 1983).
20. Mary Angela Papalaskaris and Lenhart Schubert, "Parts Inference: Closed and Semi-Closed Partitioning Graphs," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 304-309 (1981).
21. Fanya S. Montalvo, "Diagram Understanding: The Intersection of Computer Vision and Graphics," AI Memo 873, MIT (Nov. 1985).