Int. J. Man–Machine Studies (1991) **34**, 97–131

Propositional representation for graphical knowledge†

JAMES GELLER

New Jersey Institute of Technology‡, Newark, New Jersey 07102, USA

(Received 3 January 1989 and accepted in revised form 2 November 1989)

Multi-media interfaces with a graphics and a natural language component can be viewed as a Natural Language Graphics systems without a host program. We will investigate a theory of Natural Language Graphics that is based on the notion of "Graphical Deep Knowledge" defined in this research. Graphical Deep Knowledge is knowledge that can be used for display purposes as well as reasoning purposes and we describe the syntax and semantics of its constructs. This analysis covers forms, positions, attributes, parts, classes, reference frames, inheritability, etc. Part hierarchies are differentiated into three sub-types. The usefulness of inheritance along part hierarchies is demonstrated, and criticism of inheritance-based knowledge representation formalisms with a bias towards class hierarchies is derived from this finding. The presented theory has been implemented as a generator program that creates pictures from knowledge structures, and as an augmented transition network grammar that creates knowledge structures from limited natural language input. The function of the picture generation program TINA as a user interface for a circuit board maintenance system and as part of a CAD-like layout system is demonstrated.

1. Introduction

1.1. NATURAL LANGUAGE GRAPHICS

A valuable component of many systems employed in human–computer interaction is a graphics interface. Systems that combine graphics with other modes of communication such as speech or natural language are referred to as multi-modal user interfaces (Reynolds, Postel, Katz, Finn & DeSchon, 1985; Poggio, Garcia Luna Aceves, Craighill, Moran, Aguilar, Worthington & Hight, 1985). A user interface management system that incorporates a knowledge representation system as a functional part is referred to as a Knowledge Based User Interface Management System (KBUIMS) (Neches & Kaczmarek, 1986; Sullivan & Tyler, 1988). If one isolates the language and graphic capabilities of a knowledge-based multi-modal user interface from the "host program" the user interface is talking to, then one ends up with a type of program which has been known since Brown and Kwasny (1977) as a "Natural Language Graphics" program. This paper will discuss our work in Natural Language Graphics and will present an application of this work as part of a user interface. In this section we will briefly survey research that has been done

† The bulk of this research was performed at the State University of New York (SUNY) at Buffalo, but it was finished while the author spent a year at the University of Southern California at the Information Sciences Institute (USC/ISI) in Los Angeles. Partial support by the Integrated Interfaces group at USC/ISI is gratefully acknowledged.

‡ 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).

0020-7373/91/010097 + 35$03.00/0

under the name "Natural Language Graphics" or that shares enough features with NLG or KBUIMSs to be of interest to us.

The manifesto of Natural Language Graphics is a paper by Brown and Chandrasekaran (1981). Brown and Chandrasekaran supply an in depth analysis of the graphics phenomena involved in NLG. The authors discuss an implementation based on a frame representation, but the main focus of the paper is on "a design for the picture production part of the system" (p. 178).

The earliest combination of graphics with semantic networks, the knowledge representation technique that we will make use of, is reported by Giustini, Levine and Malowany (1978). Giustini *et al.* refer to the reported semantic network as "mathematical", as opposed to "linguistic", meaning that it does not span the complete set of properties that one would expect from a semantic network. Our own theory and implementation is based on SNePS, the Semantic Network Processing System (Shapiro, 1979; Shapiro & Rapaport, 1986), a fully intensional propositional semantic network.

The work by Adorni, Di Manzo and Giunchiglia (1984*a*, 1984*b*) and Di Manzo, Giunchiglia and Pino (1984) can be classified as Natural Language Graphics; however, it concentrates on problems of equilibrium, support, instantiation of unmentioned objects and space occupied by an object. It does not draw a clear distinction between the domain of diagrams and the domain of real world objects. This makes it difficult to think about different diagrammatical views of the same object. The research concentrates on natural language problems and spatial reasoning and seems to use diagrams only as a proof of the NL understanding abilities of the system, not as main subject of investigation.

On the other side, the work of Kosslyn and Shwartz (1977) on imagery describes a program that assembles a diagram from a partially propositional and partially iconic representations. This general approach (also described by Kosslyn 1980, 1981*a*, 1981*b*, 1985) captures many of the central elements of our research. However, Kosslyn and Shwartz do not present a formal catalog of representations that they use and do not investigate interactions with natural language.

Similarly, Friedell (1984) describes the generation of images from high level object specifications, but also without language interaction. He presents two example systems, one for the generation of ship images from a data base and one for the automatic synthesis of backgrounds for three-dimensional scenes.

A third approach that concentrates on image generation from a knowledge base and ignores language has been published by Zdybel, Greenfield, Yonke and Gibbons (1981) under the category "Information Presentation System" (IPS). Their AIPS system is based on the KL-ONE semantic network and is similar enough in spirit to our approach to deserve mention. However, there is disagreement concerning the success of the details of the AIPS theory. While Zdybel *et al.* (1981) state that "By an IPS we mean a system that . . . functions reasonably well without demanding custom-tooling for a particular application. . . ." (p. 978) they do not make it clear that their system lives up to this expectation. Friedell (1984) notes that "Systems such as BARAT and AIPS succeed in narrow, well-defined domains for which it is practical to provide an adequate repertoire of predefined parameteric object descriptions" (p. 54). Our own representation of attributes is very general and is not characterized by what Friedell calls "parametric object descriptions".

An important line of research that combines issues of computer vision with issues of knowledge representation is the *Mapsee*2 project (Havens & Mackworth, 1987). This system uses schemata (frame-like structures) as a knowledge representation formalism for visual knowledge. The Mapsee2 project shares the use of diagrammatic representations with our work; a hierarchical structural description of a cartographic map is produced from a hand-drawn sketch-map.

Recently Tranowski (1988) presented a system that is geared towards scene analysis, which however permits the graphical definition of simple patterns in a knowledge acquisition environment. A final line of work that is related to our own but ignores natural language interaction is pursued by Borning (1986). In his constraint based ThingLab system, a user may interact with the graphics environment through two different classes of windows, "use views" and "construction views".

Our research has grown out of work on a circuit board maintenance system, and this naturally raises the question about NLG-like work in CAD (Computer Aided Design). It turns our that references to natural language in CAD are sparse. An exception is Samad (1986) but his work deals with post processing and querying of the output of a simulation system and is not graphics oriented.

An important example of work in KBUIMS that is related to our research, is the HITS (human interface tools) system (Hollan, Miller, Rich & Wilner, 1988). HITS is heavily graphics oriented, and also has a natural language component. While the sub-systems of HITS surpass the sub-systems of our own implementation, the designers of HITS apparently had to find out that the integration of subsystems in NLG, unless planned from the beginning, is difficult, and the use of an appropriate knowledge representation system crucial. "HITS currently exists primarily as a set of independent tools, most of which are implemented on top of the Proteus knowledge base system [...]. We are in the process of reimplementing many of these tools on top of a richer knowledge representation system. . . ." (Section 4).

Finally we want to mention work (Neal & Shapiro, 1988) based on our earlier endeavors (Geller & Shapiro, 1987) which combines a semantic network with graphics and natural language processing in a user interface.

In summary, we have presented a number of different approaches that combine varying amounts of natural language processing, graphics and knowledge representation. Our own approach will rigorously describe the knowledge structures involved in a system that completely integrates graphics and natural language into the same sophisticated newtork-based propositional knowledge representation system. This description will be completed by showing the application of our theory to user interfaces.

1.2. A GLOBAL VIEW OF KNOWLEDGE REPRESENTATION

To place this research effort into a larger knowledge representation framework, four principles will be formulated that I refer to as "Shapiro's principles of AI development".

(1) If a person claims understanding of a natural language utterance or discourse, then a computer proposed to have a comparable degree of intelligence must claim understanding of the same utterance or discourse.

(2) If a person can respond to an utterance or take part in a discourse, then a

computer proposed to have a comparable degree of intelligence must be able to respond to the same utterance or take part in the same discourse in a comparable way.

(3) If a person can react to an utterance or a discourse by changing his or her behavior, then a computer proposed to have a comparable degree of intelligence must be able to react with a comparable change of behavior.

(4) The preferred way for achieving (1) to (3) is to use a fully intensional and propositional knowledge representation system.

Earlier publications by Shapiro and the members of the SNePS research group, e.g. Neal & Shapiro (1987), have concentrated on the level of intelligent behavior corresponding to (1) and (2). Recently the SNePS research program has been extended to deal with (3) on an explicit level (Kumar, 1989). However, the transition from (2) to (3) was first attacked in the work to be reported here which is based on Shapiro and Geller (1986); Geller and Shapiro (1987); and Geller (1988). The acts that have been added in this work to natural language interaction are drawing acts.

2. Graphical deep knowledge for NLG

In this core section we will discuss knowledge structures for a graphics oriented NL program. We will refer to the class of all such structures as "Graphical Deep Knowledge". More precisely we will call knowledge bases that can be used for generating diagrams "projectively adequate". In addition we want to do propositional reasoning about knowledge bases describing the physical structure of diagrams, and a knowledge base that permits doing this will be said to exhibit "deductive graphical adequacy". While the terms "visual knowledge" and "graphical knowledge" are standard AI terminology, the literature reports no good name for knowledge that exhibits projective and deductive graphical adequacy. This is why we have introduced the term "Graphical Deep Knowledge" (Geller & Shapiro, 1987).

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

2.1. THE SNePS KNOWLEDGE REPRESENTATION SYSTEM

The work presented here makes use of the SNePS Semantic Network Processing System (Shapiro, 1979; Shapiro & The SNePS Research Group, 1983; Shapiro & Rapaport, 1986) as a notational formalism as well as an implementation language. SNePS is a propositional network as opposed to an inheritance network such as KL-ONE (Brachman & Schmolze, 1985), NIKL (Robins, 1986), LOOM (MacGregor & Bates, 1987), KRYPTON (Brachman, Fikes & Levesque, 1983, 1985) or parts of KREME (Abrett & Burstein, 1987).

We will appeal to the intuition of readers unfamiliar with SNePS and present only an example instead of an in depth explanation of its syntax and semantics (which can be found in the previously given references). In Figure 1 the nodes m1, m2, m3, m4 and m5 represent propositions.† M2 represents the proposition that "D1A1" is an

† In some cases proposition nodes may be interpreted as "structured individuals".

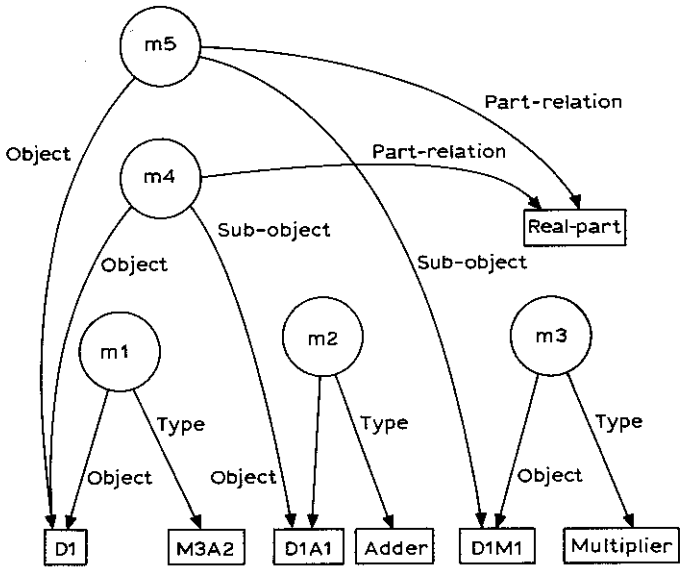

FIGURE 1. A typical SNePS network.

"Adder". The arcs "object" and "type" build a case-frame (in Fillmore's sense, 1968), i.e. the meaning carrying elements are combinations of arcs that emanate from proposition nodes. The arcs in a network point from proposition nodes to nodes representing the concepts that take part in the proposition.

M1 expresses the fact that object D1 is of type M3A2. M4 expresses the fact that the real-part relation holds between D1 and D1A1. M3 and M5 follow analogously. An equivalent first order predicate calculus representation for Figure 1 would be the following one:

type(m1, M3A2) & object(m1, D1)
type(m2, Adder) & object(m2, D1A1)
type(m3, Multiplier) & object(m3, D1M1) (1)
subject-object(m4, D1A1) & object(m4, D1) & part-relation(m4, real-part)
sub-object(m5, D1M1) & object(m5, D1) & part-relation(m5, real-part)

There is an obvious less redundant way of writing (1):

```
m1( type          M3A2
    object        D1)
m2( type          Adder
    object        D1A1)
m3( type          Multiplier
    object        D1M1)
m4( sub-object    D1A1                (2)
    object        D1
    part-relation real-part)
m5( sub-object    D1M1
    object        D1
    part-relation real-part)
```

The above five network structures exemplify two case frames:

⟨membership⟩:

type	⟨device-type⟩
object	⟨object-1⟩

(3)

⟨part-rel⟩:

sub-object	⟨object-2⟩
object	⟨object-3⟩
part-relation	real-part

(3b)

Case-frame names (before the ":") will be omitted when they seem unnecessary.

This representation might look like BNF notation, but is different because the pairs of slots and fillers may occur in any order. In continuous text, case frames are sometimes simply shown as lists of arc labels. In this notation, (3) would be represented as (type object).

We have introduced the above representational conventions to give a precise linear representation of our semantic networks. This has been prompted by Hayes (1977) who states, "If someone argues for the superiority of semantic networks over logic, he must be referring to some other property of the former than their meaning (for example, . . . their attractive appearance on a printed page)" (p. 561).

2.2. REPRESENTATIONAL CONSTRUCTS OF GRAPHICAL DEEP KNOWLEDGE

2.2.1. Form knowledge

A number of different scientific subfields and fields have been interested in the representation of forms. Among these are computer vision, computer graphics and imagery, but also solid modeling (Requicha, 1980), computer aided design (CAD), and character recognition. We argue (Geller, 1988) that no representation in any of these fields satisfies the requirements for graphical deep knowledge. These requirements are:

- The representation should be projectively adequate;
- The representation should be deductively adequate;
- The representation should be based on conceptual primitives which seem natural to the human observer;
- The representation should support relations between primitives which are natural to humans;
- The representation may contain redundant information.

To fulfill these requirements a representation that consists of basic forms (icons) and asserted relations is used. The basic forms are (supposed to be) meaningful to human observers. Every basic form is represented as a procedure that has three properties: (1) The procedure consists of calls to graphics primitives; (2) Executing a procedure of the name ⟨name⟩ results in the drawing of an object that is described by ⟨name⟩; (3) The procedure ⟨name⟩ is accessible as a concept in the knowledge representation system, i.e. it functions simultaneously as a node in a semantic network. The representation of a basic form is therefore projectively adequate and also a conceptual unit. Relations between icons are represented propositionally. A number of different proposition types is permissible which will be elaborated in this paper.

The SNePS system is used in the following way to accommodate the described form representation. The name of every basic form in the system is a base node in the SNePS semantic network. The SNePS inference machine treats it as a conceptual unit and permits reasoning about it. At the same time every SNePS node is also an uninterned LISP atom. This atom refers to a LISP function made up of calls to graphics primitives from a LISP graphics package.† Objects and forms are separate nodes, linked by an asserted proposition. This conceptual separation of forms and objects makes it possible to associate a form with a class of objects, instead of a single object.

2.2.1.1. Individual form. Figure 2 shows a structure that can be read as "the object chip1 has the form 'xand' under the modality logical". The meaning of the structure results from the combination of the form, modality and object arcs. The impact of the modality arc will be explained below. According to the notational conventions established earlier, the network in Figure 2 has the following linear representation:

```
m1( form      xand
    modality  logical                    (4)
    object    chip1)
```

The form function "xand" that draws the icon displaying an and-gate is coded as follows:

```
(defun xand (x y)
      (setq CENTER (list x y))
      (mapcar (function draw-wse)
        '((xplylnrel-wse 0 0 2  *b)                    (5)
          (xarcrel 20 -20 0  -180 *b)
          (xplylnrel-wse 20 -40 -20 0 *b)
          ...
          (xplylnrel-wse 40 -20 30 0 *b)))))
```

The function "xand" consists mostly of calls to the two graphics primitives xarcel and xplylnrel-wse. Xarcrel draws an arc and xplylnrel-wse a polyline (a train of line segments) respectively. The picture created is shown in Figure 3.

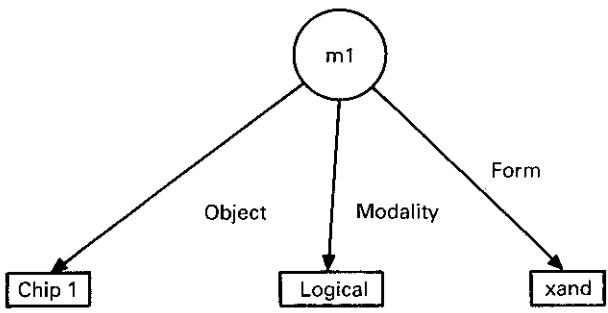

FIGURE 2. Object chip1 has the form "xand".

† The linkage of the function has been handled differently depending on the dialect of LISP used. Our favorite solution has been to use the function cell of an interned atom of the same name as the node.

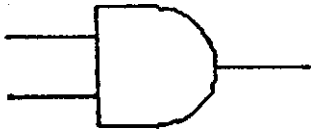

FIGURE 3. An AND gate.

Abstracting from the given example a syntactic description of an individual form would consist of the following case frame:

form	⟨form⟩
modality	⟨modality⟩
object	⟨object⟩

(6)

We will supply a descriptive semantics for every given case frame. (6) represents the proposition that the object ⟨object⟩ has the form ⟨form⟩ under the modality ⟨modality⟩.

This structure can be used to answer questions such as "What is the form of ⟨object⟩?" or for identifying all objects that have the form ⟨form⟩, or for asserting that some agent believes in the fact that ⟨object⟩ has the form ⟨form⟩. Assuming a model of an abstract graphics machine that receives the request to draw the object ⟨object⟩ under the modality ⟨modality⟩ and at a location (x, y), then the function denoted by ⟨form⟩ is applied to the arguments x and y. The location (x, y) will be the location of a privileged point of the object ⟨object⟩ called the reference point. How x and y can be determined, their meaning relative to the screen, and the nature of the reference point will all be explained below.

Two notes about the epistemic status of the syntactic variables ⟨object⟩, ⟨form⟩ and ⟨modality⟩ have to be made: (1) If one of these variables is used again later on in the context of this paper, even if it is modified by an integer number (e.g. ⟨object1⟩, ⟨form-1⟩), it refers to the same syntactic variable as defined here; (2) It has been pointed out repeatedly in the literature (e.g. McDermott, 1981) that one cannot rely on a label to express a meaning. Therefore we will not hesitate to give a semantics specification such as "⟨modality⟩ represents a modality". After all, we could have used a term such as ⟨g0001⟩ as a syntactic variable. Nevertheless we will strive to use self documenting names for syntactic variables.

The syntactic variable ⟨form⟩ stands for the concept of a form, i.e. of an entity that can be visualized by a person, and that can be projected by the abstract graphics matching; ⟨object⟩ denotes the concept of an individual object. Finally,

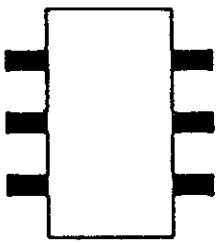

FIGURE 4. The physical structure of an AND gate.

⟨modality⟩ stands for the concept of a modality, whereby a modality can best be understood as one of several possible "views" of an object. Clearly an AND gate looks different in a logical representation (Figure 3) and a physical representaion (Figure 4).

2.2.1.2. Class form with n step inheritance. An object may inherit a form along a class hierarchy.

Syntax:

object	⟨object⟩	(7)
type	⟨class⟩	
modality	⟨modality⟩	
sub-class	⟨class⟩	(7b)
class	⟨class-2⟩	
modality	⟨modality⟩	
sub-class	⟨class-2⟩	
	:	
	:	
class	⟨class-n⟩	
modality	⟨modality⟩	
class	⟨class-n⟩	(8)
form	⟨form⟩	
modality	⟨modality⟩	

Semantics: (7) describes a simple class membership, meaning that ⟨object⟩ is a member of ⟨class⟩ under the modality ⟨modality⟩. (8) asserts that every member of the class ⟨class-n⟩ that does not have its own form has the form ⟨form⟩ under the modality ⟨modality⟩. The intermediate structures (7b) represent sub-class assertions. ⟨Class⟩ stands for the concept of a class i.e. an entity that is by itself not displayable, but which has members that are (potentially) displayable.† All other syntactic variables in the above structure have been defined in the previous sections.

Assuming that there is no other network structure present that could have an influence on the display of ⟨object⟩, we can summarize the procedural effect of the above structures in the following way. If the structure (7) is asserted, and the structure (6) is not asserted, when requested to draw the ⟨object⟩ at location (x, y), the abstract machine has to apply the function ⟨form⟩ to x and y.

A final note on forms: all forms in this investigation are assumed to be rigid.

2.2.2. Positions

2.2.2.1. Reference frames. In understanding utterances about spatial relations the identification of a correct reference frame is often the first problem that has to be solved (Sondheimer, 1976). For NLG the problem of reference frame identification means to determine whether a person is referring to a screen or a world coordinate system, and in the latter case to determine the relation between the world coordinate system and the screen coordinate system.

To project an object of the world onto a screen two conceptual steps are

† We are not interested in empty classes or infinite classes.

necessary. First a projection plane is selected, and the object is projected onto the plane with beams orthogonal to it. Then an area on the projection plane needs to be selected and mapped onto the screen or onto a window.†

A complete representation of this projection process needs to include the concepts of world coordinate systems, projection planes, screen coordinate systems and numeric values describing the details of the used projections. We will simplify our representation by permitting only a few specialized plane positions and by defining a privileged screen coordinate system. Nevertheless we will permit alternative screen coordinate systems also. We will first represent a space which is independent of a coordinate system.

Syntax:

⟨space-description⟩:
space ⟨space-descriptor⟩
space-type ⟨space-type⟩ (9)

Semantics: The space denoted by ⟨space-descriptor⟩ is of the type ⟨space-type⟩. ⟨Space-descriptor⟩ denotes any concept of a coordinate space. ⟨Space-type⟩ is one member of the set {world, plane, screen}. This definition does not make any statements about the coordinate system used in ⟨space-descriptor⟩.

Syntax:

⟨coordinate-system⟩:
space ⟨space-descriptor⟩
coord-sys ⟨coord-sys-descriptor⟩
coord-type ⟨coord-type⟩
first-axis ⟨axis⟩
second-axis ⟨axis-2⟩
third-axis ⟨axis-3⟩ (10)

Semantics: The coordinate system ⟨coord-sys-descriptor⟩ represented by an atomic node is resident in the space ⟨space-descriptor⟩, is of the coordinate type ⟨coord-type⟩ and has the three axes (⟨axis⟩ ⟨axis-2⟩ ⟨axis-3⟩) in exactly this order. ⟨Space-descriptor⟩ denotes any concept of a coordinate space. ⟨Coord-sys-descriptor⟩ denotes any concept of a coordinate system. ⟨Axis⟩, ⟨axis-2⟩, and ⟨axis-3⟩ denote concepts of coordinate axes. ⟨Coord-type⟩ is one of the atomic nodes {cartesian, polar, cylindrical, spherical}. ⟨Axis⟩ will be interpreted as an X axis, if the ⟨coord-type⟩ is cartesian, and as an R axis otherwise. ⟨Axis-2⟩ will be interpreted as a Y axis if ⟨coord-type⟩ is cartesian, and as a Φ axis otherwise. ⟨Axis-3⟩ will be interpreted as a Z axis, if ⟨coord-type⟩ is cartesian or cylindrical, ignored if it is polar, and interpreted as a τ axis otherwise. If ⟨space-descriptor⟩ is defined as a two-dimensional space by its ⟨space-description⟩, then the third axis will be ignored. We will consider the origin of the coordinate system as its reference point; therefore, any reference to the position of the coordinate system is a reference to its origin.‡

For notational purposes we will refer to world coordinate axes as X, Y and Z, to plane coordinate axes as x_p, y_p, and to screen coordinate axes as x and y (See

† It is now customary to refer to an area on the screen of a terminal as a window. Strictly speaking this is wrong, the correct term is viewport. Window refers to an area in the world.

‡ In general the reference point of an icon may be chosen arbitrarily when the icon is designed.

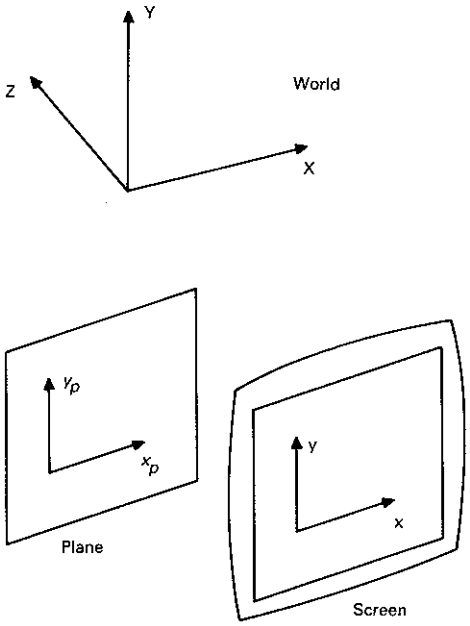

FIGURE 5. Three different coordinate systems.

Figure 5). More precisely, x and y are used generically or with reference to the privileged screen coordinate system. The discrimination between these two choices will always be possible by context.

The following positions of projection planes relative to world coordinate systems will be permitted: (1) X parallel to x_p and Y parallel to y_p; this will be called a "front view", (2) X parallel to x_p and Z parallel to y_p; this will be referred to as a "top view".

The reasons for these choices are as follows: People often look at a two-dimensional diagram with a preconception that this is really a projection of a three-dimensional world. Imagine a screen with a vertical arrangement of two circles (Figure 6). If one assumes that this scene represents a front view, then one circle is

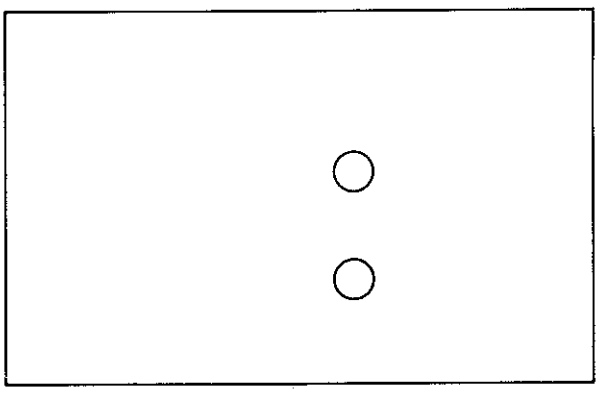

FIGURE 6. An arrangement of two circles can be described in two ways.

clearly *above* the other circle. But if a person looks at the diagram as a map and at the two circles as two trees and pictures herself "below" the lower circle, then she will think of the "upper" circle as being *behind* the lower circle! Thus, some natural language utterances can only be interpreted if one starts with the idea of a three-dimensional world coordinate system. Luckily people do not seem to assume arbitrary projection angles, so it will be sufficient for us to use (1) and (2). Interestingly, if one does not want to specify "behindness" by concrete (= numeric) terms, the world coordinate system is not necessary at all, and one can talk about one object being behind another object even in reference to a screen coordinate system!

We will now present the knowledge structure† that covers the projection choices that we have considered interesting.

Syntax:

⟨projection⟩:	
coord-sys-w	⟨coord-sys-descriptor⟩
coord-sys-p	⟨coord-sys-descriptor-2⟩
view	⟨top-or-front⟩

(11)

Semantics: The case frame (11) describes a projection. It asserts that a projection from the coordinate system ⟨coord-sys-descriptor⟩ to the (plane) coordinate system ⟨coord-sys-descriptor-2⟩ is done such that the following holds true: (1) The origins of the two coordinate systems are connected by a vector orthogonal to the plane; (2) The plane is located such that the projection amounts to a ⟨top-or-front⟩ view. The only values that can be taken on by ⟨top-or-front⟩ are "top" and "front". In other words, for a front view the projection plane is parallel to the plane defined by the [X Y] plane, and for a top view the projection plane is parallel to the plane defined by the [X Z] plane. It is permissible to omit the slot containing ⟨coord-sys-descriptor⟩. In that case the ⟨view⟩ structure is used only to interpret fuzzy natural language terms.

Without such an assertion the following question would be meaningless to the system: "Is this a top view or a front view?"

The above structure has the following effect on display requests to the abstract graphics machine. If required to display an object A "behind" an object B, and ⟨top-or-front⟩ is "top", then A will be displayed vertically above B. If the view specified is "front" then A will be displayed at the same location as B, such that B is drawn later and overdraws A. "In-front", "above" and "below" are interpreted analogously.

The reader might think that this representation is too parsimonious to be useful, because we have not mentioned anything about numerical values concerning the mapping from the projection plane to the screen. However, this is often not the case, because an NLG user has different expectations from users of standard graphics packages. An NLG user prefers to specify the *objects* that he wants to see and expects the program to find its own window that includes all these objects as large as possible, and to map this window correctly into a user selected viewport. The necessary translation factor and the necessary shift vector are dynamically

† This is somewhat simplified compared with the representation in Geller (1988).

adapted whenever necessary. For more details on possible projections and for the question of how to represent units in a coordinate system we have to refer the reader to Geller (1988). There we discuss concrete units (pixels) as well as fuzzy units (near/far).

2.2.2.2. The representation of positions. Positions are the most complex phenomena in graphical deep knowledge. Nevertheless it turns out that one can get by with a single structure to represent every possible case. In this section this overarching representational structure will be discussed. First we have to introduce the representation of a measurement which consists of a value and a unit.

Syntax:

⟨linear-measure⟩:
value ⟨value⟩ (12)
unit ⟨unit⟩

Semantics: ⟨Linear-measure⟩ describes a measurement, such that ⟨value⟩ is the value and ⟨unit⟩ is the unit of the measurement. ⟨Value⟩ may be any numerical value or an element of a small set of conceptual values, namely {left, right, above, below, near, far, behind, in-front}. We will refer to numerical values as "concrete", and to conceptual values as "fuzzy". ⟨Unit⟩ may be any unit of length measurement, including one of a small set of conceptual units, namely {left-right, above-below, near-far, behind-front}, or it may be the concept of an object used as a unit.

Syntax:

⟨angle-measure⟩:
value ⟨angle-value⟩ (13)
unit ⟨angle-unit⟩

Semantics: An ⟨angle-measure⟩ consists of an ⟨angle-value⟩ and an ⟨angle-unit⟩, such that ⟨angle-value⟩ is any number† and ⟨angle-unit⟩ is any measuring unit for angles, such as degrees or radians.

In the specification of a position we will make use of linear as well as angular measures; we define therefore the following alternative. The exclamation mark denotes a BNF-like "or", but remember that this and all previously shown structures are *not* BNF structures, because slot-filler pairs may be arranged in any desired order.

Syntax:

⟨measure⟩:
⟨angle-measure⟩ ! ⟨linear-measure⟩ (14)

It is sometimes useful to have an area measure, so we include it here.

Syntax:

⟨area-measure⟩:
value ⟨area-value⟩ (15)
unit ⟨area-unit⟩

† More precisely ⟨angle-value⟩ stands for a node that represents the concept of a number that can specify an angle. We will somewhat relax the rigor of our specifications.

Semantics: An ⟨area-measure⟩ consists of an ⟨area-value⟩ and an ⟨area-unit⟩, such that ⟨area-value⟩ is any number or one of the fuzzy values {small, large}, and ⟨area-unit⟩ is any concrete measuring unit for areas or the conceptual unit small-large. We now need to assign a direction to a measurement.

Syntax:

⟨component⟩:
direction ⟨axis⟩
measure ⟨measure⟩ (16)

Semantics: ⟨Component⟩ describes an oriented measure, such that ⟨axis⟩ is an axis of a known coordinate system, and ⟨measure⟩ is a measure as defined previously. ⟨Axis⟩ may refer to a linear axis, as well as to a conceptually bent axis, as they are made use of in non-cartesian coordinate systems.

Syntax:

⟨vector⟩:
coord-sys ⟨coord-sys-descriptor⟩
component ⟨component⟩
component ⟨component-1⟩
component ⟨component-2⟩ (17)

Semantics: (17) describes a ⟨vector⟩ that is defined in a coordinate system ⟨coord-sys-descriptor⟩ and that consists of three components ⟨component⟩, ⟨component-1⟩ and ⟨component-2⟩ which are all ⟨component⟩s. In the most general case this vector will be defined in a three-dimensional space. However, as noted before, it is permissible to omit irrelevant or unavailable information, therefore a two-dimensional vector will be represented by omitting the slot for ⟨component-2⟩.

Syntax:

⟨object-position⟩:
object ⟨object⟩
relpos ⟨vector⟩
rel-to ⟨object-or-co⟩
modality ⟨modality⟩ (18)

Semantics: ⟨Object⟩ denotes an object concept. ⟨Object-or-co⟩ denotes an object concept or a coordinate system concept. ⟨Object-position⟩ describes the position of an object ⟨object⟩ by supplying a reference object ⟨object-or-co⟩ and a vector ⟨vector⟩ that has its starting point in the reference point of the reference object ⟨object-or-co⟩ and its end point in the reference point of ⟨object⟩. The ⟨object-position⟩ specified this way is valid for the modality ⟨modality⟩ only.

Syntax:

⟨class-position⟩:
class ⟨class⟩
relpos ⟨vector⟩
rel-to ⟨object-or-to⟩
modality ⟨modality⟩

Semantics: The ⟨class-position⟩ case frame defined in (19) defines the relative position for a class ⟨class⟩. Every ⟨object⟩ that is a member of ⟨class⟩ and that does not own a position by virtue of an ⟨object-position⟩ case frame and that does not inherit a position from a sub-class of ⟨class⟩ has its position assigned by the ⟨class-position⟩ case frame. The position is specified by the vector ⟨vector⟩ positioned such that it starts in the reference point of ⟨object-or-co⟩ and ends in the reference point of ⟨object⟩. The ⟨class-position⟩ is only valid for the modality ⟨modality⟩.

Syntax:

⟨position⟩:
⟨class-position⟩ ! ⟨object-position⟩ (20)

Semantics: A position description ⟨position⟩ is either a ⟨class-position⟩ or an ⟨object-position⟩. We will now represent an ⟨object-position⟩ in its expanded form by replacing syntactic variables by appropriate sub-structures.

⟨object-position⟩:

object	⟨object⟩				
relpos	coord-sys	⟨coord-sys-descriptor⟩			
	component	direction	⟨axis⟩		
		measure	value	⟨value⟩	
			unit	⟨unit⟩	
	component	direction	⟨axis-2⟩		
		measure	value	⟨value-2⟩	
			unit	⟨unit-2⟩	
	component	direction	⟨axis-3⟩		
		measure	value	⟨value-3⟩	
			unit	⟨unit-3⟩	
rel-to	⟨object-or-co⟩				
modality	⟨modality⟩				

(21)

(21) represents a proposition that, in modality ⟨modality⟩, ⟨object⟩ is ⟨value⟩ ⟨unit⟩s in the ⟨axis⟩ direction and ⟨value-2⟩ ⟨unit-2⟩s in the ⟨axis-2⟩ direction and ⟨value-3⟩ ⟨unit-3⟩s in the ⟨axis-3⟩ direction away from the position of ⟨object-or-co⟩ in the coordinate system given by ⟨coord-sys-descriptor⟩.

Given a display request for ⟨object⟩ in the modality ⟨modality⟩, ⟨object-position⟩ can be used to derive the position of ⟨object⟩, if the position of ⟨object-or-co⟩ and the valid reference-frame ⟨coord-sys-descriptor⟩ are known.

Above representational structure for positions is flexible enough to represent all the distinctions that we need to capture. In Geller (1988) we demonstrate this in depth. Here we will limit ourselves to name the distinctions which can be expressed.

- Concrete (numeric) *vs* fuzzy relative position descriptions
- Cartesian *vs* polar coordinates
- Absolute *vs* relative positions
- Positions with explicit *vs* deduced reference objects
- Own *vs* inherited relative positions
- Absolute coordinate units *vs* reference objects used as coordinate units
- Screen *vs* plane coordinates
- Real three-dimensional coordinates *vs* two and a half-dimensional representations

2.2.3. The representation of attributes in Graphical Deep Knowledge

2.2.3.1. Types of attributes. One of the big advantages of a knowledge-based graphics system is that one can freely and dynamically associate object attributes and pictorial attributes. Besides analysing attributes into these two classes (as done in detail in Geller, 1988) we can analyse attributes according to the number of *attribute values.* Although there are very few clear cut cases, it seems reasonable to represent faultiness of a device as a binary attribute. The device is either in need of repair or it isn't.† In other words, this is an attribute that does not have any attribute values.

An attribute such as color can better be captured by an attribute value from a small base set. Although one could deal with 10 binary attributes of the form *being-red vs* not being-red, etc. it seems more natural to talk about the attribute color which can have one of 10 attribute values. We will refer to the attribute itself as *attribute class* and to the *attribute value* as such or as *argument position.* The semanticist Lyons (1977) refers to this distinction as "bipartite sense-components consisting of (i) a superordinate marker taken from the set M = {SEX, COLOUR, AGE, SPECIES, . . .} and (ii) a subordinate marker μ, specifying which particular location within the domain denoted by the superordinate marker is denoted by the subordinate marker" (p. 325). The analog extension to two, three or more attribute values presents no difficulty.

If one combines attributes with forms then two ways of interpreting the relation between these two entities become necessary in the presence of an abstract graphics machine. An attribute may be strictly descriptive, i.e. describe a feature that is incorporated in the form of an object. We will refer to such attributes as *unmapped* attributes. Alternatively an attribute might be used "modificatory", to change the form of an object which has been inherited along a class hierarchy. This will result in a modified graphical display. We will refer to such an attribute as a *mapped* attribute. For a discussion of unmapped attributes such as "symmetry", refer to Geller (1988).

The final way to discriminate between different attributes which we want to discuss, is the distinction between *relative* attributes and *absolute* attributes. An attribute such as "faultiness" of a device is absolute. It is possible to decide that the device is faulty, without any reference to another device. On the other hand, an attribute such as "large" requires a reference class to be meaningful.

2.2.3.2. Simple attribute representations. We will initiate this section with the representation of an absolute attribute with one argument position.

Syntax:

⟨a-attribute-1⟩:
atrb-cls attribute-class⟩ (22)
atrb ⟨attribute-value⟩

Semantics: (22) is a structured individual describing an absolute attribute with one argument position, whereby ⟨attribute-class⟩ denotes any concept of an attribute, and ⟨attribute-value⟩ denotes any concept of an attribute value belonging

† Even if the state of a device can be described by a number of different possible levels of functioning, we will assume that in every case a decision has been made whether a repair action is needed or not. This decision defines a binary faultiness attribute.

to this ⟨attribute-class⟩. Here and in all future uses of ⟨attribute-value⟩ it is permissible that ⟨attribute-value⟩ be itself a structured individual. The case-frame for an absolute object with two argument positions becomes by simple extension (atrb-cls atrb1 atrb2).

The structure for an attribute with three argument positions follows analogously. The structure for a zero-value attribute ⟨a-attribute-0⟩ is created by omitting the ⟨attribute-value⟩ slot from the ⟨a-attribute-1⟩ definition. With this we can define the structure of an absolute attribute ⟨a-attribute⟩ as a choice.

⟨a-attribute⟩:
⟨a-attribute-0⟩ ! ⟨a-attribute-1⟩ ! ⟨a-attribute-2⟩ ! ⟨a-attribute-3⟩ (23)

We will now turn to the representation of relative attributes.

Syntax:

⟨r-attribute-1⟩:

rel-atrb-cls	⟨attribute-class⟩
atrb	⟨attribute-value⟩
ref-set	⟨reference-set⟩

(24)

Semantics: (24) is a structured individual describing a relative attribute with one argument position, whereby ⟨attribute-class⟩ denotes any concept of an attribute, and ⟨attribute-value⟩ denotes any concept of an attribute value belonging to this ⟨attribute-class⟩. ⟨Reference-set⟩ denotes any concept of a group of objects, being either a ⟨class⟩ of objects, or a structured object with parts. If an ⟨r-attribute-1⟩ case frame is given without a ⟨reference-set⟩, then it is assumed that the immediate super-class of the object provides the reference. (This relies on a non-tangled class hierarchy).

If somebody says, "Joe is tall", we assume that Joe is a human, and that his height is probably in the area of 6 feet and over. Therefore we will not represent the reference class explicitly in the system. If somebody wants to preempt the default interpretation he has to supply a reference set. This would be the case in a sentence such as "Joe is tall for a kid of four years", or "Joe is small for a basketball player".

The extension of this representation to two, three or zero attribute values raises no problems, and we can summarize an attribute as being either a relative attribute or an absolute attribute.

⟨r-attribute⟩:
⟨r-attribute-0⟩ ! ⟨r-attribute-1⟩ ! ⟨r-attribute-2⟩ ! ⟨r-attribute-3⟩ (25)

⟨attribute⟩:
⟨r-attribute⟩ ! ⟨a-attribute⟩ (26)

After clarifying what an ⟨attribute⟩ is, we now have to assign it to an entity in our knowledge base.

Syntax:

⟨attribute-assignment⟩:

patient	⟨patient⟩
attr	⟨attribute⟩
modality	⟨modality⟩

(27)

Semantics: The ⟨attribute-assignment⟩ case frame asserts that a patient⟨patient⟩ has an attribute⟨attribute⟩ in the modality ⟨modality⟩. ⟨Patient⟩ represents the syntactic class of all units that may receive an attribute. It may be an ⟨object⟩, a ⟨form⟩ or a ⟨class⟩. This structure makes it possible to query what attributes ⟨patient⟩ has.

2.2.3.3. Attribute mappings. One of the attractive features of GDK based systems is that mappings between different representational formats can be done declaratively and therefore changed easily, but still show a procedural effect. This has a number of practical applications. Color is usually a strong medium of communication and can be used to symbolically represent other attributes which are of a non-graphical nature. However, if a specific user happens to be color-blind, then he will lose important features of the system. In a knowledge-based representation system one can specify the desired mapping of object attributes to picture attributes explicitly. Because this information is accessible to the abstract graphics machine, it can be used to change the display format.

We first need to introduce a structured individual that will be used in attribute mappings.

Syntax:

⟨value-mapping⟩:
expressed ⟨attribute-value⟩
expressed-by ⟨argument⟩ (28)

Semantics: (28) expresses a structured individual describing a mapping between one ⟨attribute-value⟩ and a corresponding argument to a modifier function ⟨argument⟩. The ⟨attribute-value⟩ usually describes an invisible attribute value, while the ⟨argument⟩ is its corresponding symbolic attribute value. ⟨Argument⟩ must be represented by a base-node. The following structure completely describes how to represent the necessary mapping.

Syntax:

⟨attribute-mapping⟩:
attr ⟨attribute-class⟩
mod-func ⟨modifier-function⟩
val1 ⟨value-mapping⟩
val1 ⟨value-mapping⟩
val1 ⟨value-mapping-2⟩
:
:
modality ⟨modality⟩ (29)

Semantics: The structure (29) expresses the proposition that attributes of the class ⟨attribute-class⟩ can be expressed graphically by the functional ⟨modifier-function⟩. In addition it expresses the correct mapping between argument positions of the attribute case frame and arguments of the ⟨modifier-function⟩. Each syntactic variable named ⟨value-mapping⟩, possibly with a number after the word "mapping", expresses one mapping between an argument position and a corresponding function argument. The "val1" arc makes it possible to differentiate between attributes with one or more argument positions. If an attribute happens to have two argument positions then the ⟨value-mapping⟩ case frames are extended to have

"val2" arcs. The "val2" arcs mark mappings for the second additional argument. For attributes with three values "val3" arcs are necessary.

This structure incorporates the knowledge necessary to answer questions such as "How would you display something ⟨attribute-value⟩?" The procedural effect of a mapped attribute comes about in the following way. Assume that the user's request is to display an object which has the attribute ⟨attribute-value⟩ belonging to the attribute class ⟨attribute-class⟩ and that ⟨attribute-class⟩ is linked to the LISP function ⟨modifier-function⟩ by a structure such as (29). Assume further that the ⟨attribute-value⟩ is linked to ⟨first-argument⟩. The abstract graphics machine then has to call the function ⟨modifier-function⟩ with the form of the object as first argument and with ⟨first-argument⟩ as second (first additional) argument. So the ⟨modifier-function⟩ takes a ⟨form⟩ as the main argument, and it returns a form-function that has been changed such that if it is executed it will incorporate the attribute expressed by ⟨attribute-value⟩.

Example:

```
m14( attr        state
     modality    logical
     mod-func    tint-jg
     val1        m12(  expressed      faulty                (30)
                       expressed-by   green)
     val1        m13(  expressed      working
                       expressed-by   magenta))
```

The above example expresses that faultiness of the attribute-class "state" is expressed by the "tint-jg" function with the argument "green" while a "state" of "working" is expressed by the color "magenta".

The use of the modifier function requires some more clarification. In our theory an attribute-class is considered as an abstract functional which takes the object it is applied to as argument. If there are any attribute-values asserted, then these correspond to additional arguments to be supplied to the functional. The node at the end of the "mod-func" arc is at the same time the concept that represents the abstract functional, and a reference to the code that performs the operations of the concrete functional. This is done in complete analogy to form concepts which simultaneously refer to procedures embodying graphics code.

This attribute theory is true to the following two important principles:

(1) A form which is modified by an attribute is itself a form and should therefore be represented consistently with all other form representations.

(2) All attributes in the system should be treated consistently. The other way one could incorporate attributes in a form function is to make the attribute-values arguments to the form functions themselves. However, this would require the user to predict *all* attributes he would ever want to use, or to recode all forms for every new attribute, or to use the method of modifier functions for all the attributes he had not thought of in the first place. This would create form functions with *many* arguments in the first place and would still result in *ad hoc* extensions for newly discovered attributes. Obviously none of these alternatives is very satisfying.

In conclusion, the solution to the previously raised problem of changing the representation of a certain attribute by color to a representation by a special line style is to introduce a new attribute mapping.

2.2.3.4. Inheritability of attributes. Some observations on inheritance and inheritability will be offered in the section on part hierarchies. In this section it will simply be pointed out that attributes might or might not be inheritable, and that this item of information is dependent on the attribute itself and can be communicated in a simple sentence. Therefore it should be representable in the system in a simple declarative structure.

Syntax:

inheritable ⟨attribute-class⟩ (31)

Semantics: The above structure expresses the assertion that ⟨attribute-class⟩ is an inheritable attribute.

It will be explained later that if an object has attributes such that their ⟨attribute-class⟩es are *not* marked by the inheritable case frame, then they are *not* considered for being propagated down a (part) hierarchy. If an attribute class is marked by structure (31) then all attributes will be applied not only to objects for which they are asserted, but also to all their parts.

2.2.4. Part hierarchies

Part hierarchies have been of fundamental importance in a number of different areas of artificial intelligence. Knowledge representation (Papalaskaris & Schubert, 1981) has dealt with them as well as hardware modeling in maintenance (Taie, 1987) and research in computer vision (e.g. Leyton 1986; Biederman, 1987).

Our interest in part hierarchies is motivated by the need for a method to decide *what content* to put on the screen of an NLG system and *how to organize it* to be optimally useful to a viewer. In KBUIMS (knowledge-based user interface management system) design, this complex of problems has been referred to as "presentation planning" (Arens, Miller & Sondheimer, 1988).

Part hierarchies permit a strategy to decide what to show and how to avoid information overload: "do not show all the parts of a requested object". If a simple display is expected, "just show an integral object". If a more informative display is expected, "then show the integral object with its parts". More generally, "control the complexity of a display by selecting the number of levels of the part hierarchy that are shown on the screen". This method has led us to distinguish three different types of part hierarchies (Geller and Shapiro, 1987) which we will discuss in detail in this report. An alternative analysis of part-whole relations based on sentence patterns has been described by Winston, Chaffin and Herrmann (1987). A comparison between their analysis and our analysis is contained in Geller (1988) and is also the subject of a forthcoming paper.

2.2.4.1. The definition of parts. It is difficult to find a necessary and sufficient condition for the part relation, but we can improve the definition given by Winston *et al.* (1987) who characterize being a part by "the elements of inclusion and connection" (p. 438). In addition to this we think of parts as being smaller than their corresponding wholes.

For abstract objects it is necessary to talk in terms of a mapping function that transforms them such that a spatial measuring function or a counting function can be applied to the image. Such a mapping function always selects a salient feature that is

common to both whole and parts. We will formulate this in the following necessary condition for being a part.

If an object P is part of another object W, then a mapping m and a function f exist such that m and f fulfil the following four conditions: (1) m is a mapping function that can be applied to both P and W and which transforms the same salient feature of both of them into a spatial or into a countable representation; (2) f is a measuring function which assigns a measure of size or a count to the mapping of W as well as to the mapping of P; (3) The size assigned by f to the mapping of W must be larger than the size assigned by f to the mapping of P.

$$f(m(P)) < f(m(W)) \tag{32}$$

(4) The sum of the measures or counts of the mappings of all *immediate* parts of an integral object is smaller or equal to the corresponding measure or count of the mapping of this object. If there are k immediate parts this can be formulated as follows.

$$\sum_{i=1}^{i=k} f(m(P_i)) \leq f(m(W)) \tag{33}$$

The reason for the use of the sign $\leq$ will be explained with an example. If we consider a forest and apply a count function f, then the sum of trees will be exactly equal to the number characterizing the forest. If we use an area function instead, the sum of the areas of the trees will be smaller than the area taken by the whole forest.

2.2.4.2. A general purpose part representation. We will now introduce an overarching representational structure that covers the three different types of part hierarchies that we have found necessary to distinguish.

Syntax:

object	⟨object⟩
modality	⟨modality⟩
part-relation	⟨real-assem-clu⟩
sub-object	⟨object-2⟩

(34)

Semantics: The above structure asserts that ⟨object-2⟩ stands in the part-relation to ⟨object⟩ which is specified by ⟨real-assem-clu⟩ and is valid in the modality ⟨modality⟩. ⟨Real-assem-clu⟩ may be one of the base nodes {real-part, sub-assembly, or sub-cluster}.

2.2.4.3. Real parts. In the realm of graphical deep knowledge an object P is defined to be a "real" part of another object W if the following conditions are fulfilled:

(1) The object P is considered part of the object W in the real world;

(2) Both the object W as well as the object P have a form, i.e. they are both by themselves displayable and can be displayed simultaneously; and

(3) The display of W without the display of P creates a "useful" image.

The last criterion is of course in the eye of the beholder and cannot be formalized any further. The purpose of this definition is obvious: for real parts the complexity of a display can be limited by showing only the integral object without its parts, and without in this way creating an "amputee". For this purpose we assume that the

form of W shows some sort of sketch that is suggestive of the object and all its parts without showing the parts in too much detail.

The representation of a real part is done by placing the concept "real-part" in the position marked by the arc "part-relation". If requested to display an object with real parts at a low level of graphical complexity, then this is interpreted as a requirement to display it alone. If required to display it with complete information, then also its (first level) real parts will be displayed, although they *were not explicitly requested.*

2.2.4.4. Assemblies. In the realm of GDK an object P is defined to be a "sub-assembly" of another object W if the following conditions are fulfilled:

(1) The object P is considered part of the object W in the real world;

(2) Both the object W as well as the object P have a form, i.e. they are both by themselves displayable and can be displayed simultaneously.

(3) The display of W without the display of P creates an image which is not desirable for the user.

An object that has sub-assemblies, such as W in the above definition, is called an "assembly". An assembly may also have real parts. An example from the domain of circuit board maintenance will clarify the changed condition. In a purely functional representation a pin of an integrated circuit will not be displayed at all, it will just be implied by the wire that connects the pin to another circuit. In maintenance such a display is not sufficient, because the pin as well as the connecting wire can be defective. Therefore one would want to display pin and wire separately, and this is usually done by a little rectangle, representing the pin and called the *port* of the component.

It would make no sense to display a component and the wire leading to it, but omit the port that creates the connection between these two objects. In other words, whenever a display of the component is required, one automatically also wants the ports shown. A representation of ports as real parts of the component is therefore not advisable, because a user could specify a complexity limited display which would show the component but omit its parts. Therefore the ports have to be made sub-assemblies of the component.

For display purposes the sub-assembly structure in a sense overrides user requests for low complexity displays. If the display of an object is requested, then its sub-assemblies will be displayed, even if the user has asked for a display of low complexity. Sub-assemblies are non-separable parts.

2.2.4.5. Clusters. In the realm of GDK an object P is defined to be a "sub-cluster" of another object W if the following conditions are fulfilled:

(1) The object P is considered part of the object W in the real world.

(2) The object P has a form, while the object W usually has no form at all in the real world but has an assigned *symbolic* form.

(3) It is not desirable to display P and W together.

An object, such as W in the above definition, that has sub-clusters is called a "cluster". A cluster may not have real parts or sub-assemblies, although some of the sub-clusters may be real parts of another object. Several sub-clusters are said to "form a cluster" if they are all and the only subclusters of a cluster. Clusters are standing nearer to what one could call an abstraction hierarchy, than the other two types of part hierarchies. The definition given here corresponds to an improvement of the definition of clusters presented in (Geller & Shapiro, 1987). In our original

definition symbolic forms were always only boxes, and this fact was not represented in the knowledge base. This is quite satisfying for technical applications, however, if one wants to replace a cluster of trees in a map by a green blob, then this blob will not usually be a rectangle. This improved definition permits one to associate a symbolic form with every super-cluster, as a "real" form is associated with every real super-part, and store it in the knowledge base.

If the drawing of a cluster is requested in complete detail, then only its sub-clusters will be drawn. If its drawing is requested at an appropriately reduced complexity level, then such a request is interpreted as a command to display the symbolic form of the cluster itself, but not its sub-clusters.

It is permissible to have an object with real parts, such that some of those parts form one or more clusters. The usefulness and naturalness of such a construct is discussed in (Geller, 1988).

2.2.4.6. The inheritability of attributes in a part hierarchy. Consider the attribute of "faultiness", which is very popular in the domain of circuit board maintenance. At the beginning of a maintenance session it is known that a whole board is faulty. The purpose of the maintenance session is to narrow down the fault to a single, or possibly a few, faulty components and acquit all the others. To inherit the attribute of faultiness would completely defeat the purpose of the maintenance system!

However, in creating a picture from predefined components, another view emerges. It is the nature of (our) graphic primitives that they do not represent an object in a size invariant format. In other words, the form-function of an object always incorporates a size. To assert in the network that one wants a scaled multiplier, this operation has to be represented as an *attribute* of the picture of the multiplier.

If a whole board is to be scaled, then all of its parts also have to be scaled. In that case it is obvious that one wants to inherit an attribute from an object to its parts. So, two attributes have been shown, an inheritable one, and an non-inheritable one. This is the reason why inheritability of an attribute has to be asserted explicitly. Also note again that this is inheritance along a *part* hierarchy, not along a class hierarchy, a technique that has not been used extensively in the KR literature. [We know of one approach that permits inheritance to any "grouping" (Smith, 1983) including parts.] The case frame for asserting inheritability has been presented in the section on attributes.

The necessity of inheritance along part hierarchies is a very important and interesting finding, because it forces us to raise some criticism of the KL-ONE family of knowledge representation (Brachman & Schmolze, 1985). KL-ONE and its descendents comprise the currently most popular family of network based knowledge representation systems in the field. The basic assumption made by these systems is that knowledge representation environments should include a taxonomic reasoner that is operating on a class hierarchy (IS-A hierarchy). The KL-ONE interpreter automatically takes care of inheritance (better: role inheritance) along this class hierarchy. However, it does not supply a general purpose inheritance mechanism for other hierarchies, such as part or containment hierarchies. As we have shown, it is a reasonable request to ask for inheritance along part hierarchies, and it seems that a knowledge representation system should treat different inheritance hierarchies consistently.

SNePS, a propositional network-based knowledge representation system, does not

supply any automatic inheritance, but supplies the ability to write path-based inference rules. An interpreter for such rules is implemented. It is the responsibility of the user to incorporate any required inheritance in his specific network interpreter. This is consistent with SNePS' status as a network at what Brachman has called the "logical level" (Brachman, 1979). However, knowledge representation systems at the "epistemic level" (such as KL-ONE) should give due consideration to uniform treatment of other major ontological hierarchies.

2.2.5. The class hierarchy

In our theory a non-tangled class hierarchy is used for standard downward inheritance (Geller, 1988). However, we also supply a limited upward inheritance facility. We find justification for this in the psychological research on categorization. The cognitive science literature reports three different approaches to categorization (Smith & Medin, 1981) the *classical* approach, the *prototype* approach and the *exemplar* approach. The classical approach has been all but totally rejected from a cognitive point of view. It requires that every member of a class be described by necessary and sufficient conditions.

The protype view as developed by Eleanor Rosch (1978) describes a "protoype" as a summary description of all the members of a class. The third theory of categorization, the exemplar view, differs from prototype theory in the following way. The summary description used by prototype theory is not necessarily identical to any existing member of the category. Exemplar theory on the other hand postulates the use of one or more stored real exemplars of the category; in other words *no* summary description exists.

The exemplar view of categorization permits us to think in terms of upward inheritance from an individual to a class, because if we do not assume a summary description we may not associate attributes with it, and then the only source from which to derive inherited attributes are other exemplars. This implies that it must be possible to inherit attributes from one exemplar upwards to a class and then back downwards to another exemplar.

For example, a knowledge base in our system might contain an object with no specified form that belongs to a class hierarchy. Classical downward inheritance would search up in the hierarchy until at some higher level a form is encountered. However, it might happen that no form is found anywhere in the hierarchy. In our interpretation of the exemplar theory it is valid to do a *down* search in the hierarchy for an object that belongs to the same class as the current focus object, and to inherit an existing form with *up-and-down inheritance* for it.

The idea of up-inheritance is not popular in AI. It is either ignored or explicitly prohibited. For instance, knowledge representation of the NETL style (Fahlman, 1979), which is based on marker passing, prohibits the idea of inheritance according to an up-and-down-movement because if one would permit markers to move up and down in the network the whole network would eventually be marked.

One is tempted to interpret up-and-down inheritance by considering the first step (the up-inheritance) as a form of generalization or inductive learning. However, this is not what we have in mind, because the representation of the class itself is *not* changed by a step of up-and-down inheritance. If a class should have many members only one of which has a form, and if this form should be changed after one

application of up-and-down inheritance, then the second application of this inheritance rule will supply the new form, not the old form. If we were talking about a step of generalization, then the class would preserve the form after the first use of up-inheritance.†

Are we then making a decision for the universal use of examplar inheritance and against prototype theory? Clearly this is not our intention, because up-and-down inheritance is *only* used when no sufficient information is associated with the classes used for inheritance, i.e. when our version of a summary description fails. We do not eliminate the use of a summary description!

For practical purposes we have limited the use of up-and-down inheritance in two ways. Up-search is done from the lowest level, the level of the individuals, to the level immediately above it, i.e. to the lowest level of classes. If there is no other member in this class, or if the other members do not carry the desired information, then up-and-down inheritance fails. One can argue that this does not make complete use of the class hierarchy, but it seems like a reasonable compromise, because humans use hierarchies that are flat and bushy. Rosenfeld has even argued that it is not necessary to view operations on hierarchies as recursive to an arbitrary depth, because this constitutes an unnecessary effort if one has only a flat hierarchy.‡

Secondly, up-and-down inheritance is used only for information that is urgently needed, and not as the default case. In a graphics system the one item of information that is obviously needed most is the form of an object, for which no "reasonable defaults" can be supplied. We will now formally define up-and-down inheritance which we also refer to as exemplar inheritance.

Definition: Exemplar inheritance. If an individual is missing information about an important property, and this property cannot be derived by inheritance from a superclass of the individual, then the property may be inherited from any of the other members of the *immediate* superclass of the individual.

In our domain only "forms" are considered important, and we have therefore decided not to represent the fact that a property is important by an explicit assertion.

It is not yet clear what happens when several members of a class offer different properties for upward inheritance. In such a case a combined strategy of majority and recency may be used. In addition we argue (Geller, 1988) that this does not constitute a real problem in the GDK domain.

3. Reasoning

The major reason for introducing the notion of graphical deep knowledge as separate from graphical knowledge has been the interest in doing reasoning about graphical structures. The first step of making a corpus of representations accessible to logic based reasoning is to transform it into a well formed declarative format with a defined syntax and semantics. It has been the approach of this investigation to limit the procedural representations which at some point are not avoidable in graphics to a small area, namely to iconic primitives. All conceptual relations between these iconic primitives are represented declaratively.

† This is not necessarily true if one wants to deal with generalization combined with truth maintenance.

‡ Azriel Rosenfield, talk 4/28/87 SUNY at Buffalo, on "Recognizing unexpected objects".

The second step is to formally define reasoning patterns. SNePS provides two different facilities for doing so, a system of rules and a system for defining paths. Although the rules that can be defined are very powerful and permit quantification as well as the use of non-standard connectives (Shapiro, 1983) we have chosen to concentrate in our implementation on the use of paths which are more efficient.

Path-based inference in SNePS assumes that one has a node of a well-specified category available (typically an "object") and follows the arcs that are pointing to this node backwards until one hits a node describing unknown and interesting information (for instance a "form" or one coordinate of a position). The well specified case frames of GDK assure that if the required information exists at all in the network, then it will be reachable by a well defined path. Paths can be described by a LISP-like language as explained in Shapiro *et al.* (1983), but we will limit ourselves to an example.

```
(find                                                          (35)
    compose
      form-!
      (domain-restrict
        (modality function)
        class)
      (kstar
        (compose
          class-!
          (domain-restrict
            (modality function)
            sub-class)))
      type-!
      (domain-restrict
        (modality function)
        object))
      pcm-1)
```

The "find" function denotes a retrieval operation from a SNePS network. Two arguments are supplied to this operation, a modality (namely "function") which is used at three positions in the find call, and an object, namely "pcm-1". All other symbols in (35) correspond either to arcs of GDK case frames or to keywords of the path language. The goal of this operation is to retrieve a form for pcm-1. Figure 7 will be helpful in understanding this example.

The "compose" keyword introduces a path. A path is constructed from its tail to its head; therefore, we start with a "form-" arc that would emanate from our requested result, should it exist. (Every arc A in the system has an antiparallel arc that is not shown in the diagram and that has a name of the form A-). The exclamation mark indicates that we are looking for an assertion, i.e. it would not be sufficient to find a correct structure if it only described a hypothesized proposition. The "(domain-restrict (modality function) . . ." piece expresses an additional constraint on this assertion: it must dominate a node "function" by way of a "modality" arc. Finally our path leaves the assertion node along a class arc. This time no negation is specified, because the path actually runs parallel to the arc.

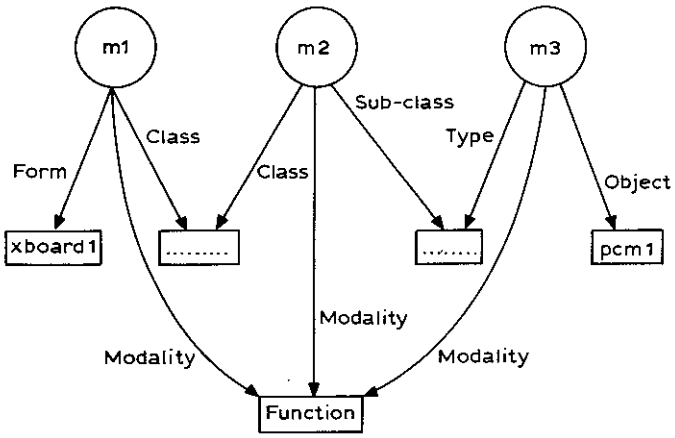

FIGURE 7. An example structure for path-based inference.

The "kstar" keyword indicates that the structure that is parenthesized together with it can be repeated 0, 1 or arbitrary many times, i.e. it is a Kleene * operator. The potentially omitted or repeated structure consists of a "class-" arc, followed by a "sub-class" arc. As before, this piece of path can be passed only if the node dominating it is asserted ("!"), and if it is also dominating a modality arc pointing to a "function" node. Finally the structure must terminate with a "type" arc which is traversed in the reverse direction, and an "object" arc that coincides with the head of the path which is pointing to "pcm-1". Like before, modality and assertional status must be correct, otherwise the path traversal fails.

For use in a graphical generator function the described path is encapsulated in a LISP function, with "pcm-1" and "function" replaced by parameters. For answering a question about an object the same path will be used for information retrieval.

4. Implementation

We will now discuss the implementation of the theory described in the previous sections which was realized at USC/ISI (the Information Sciences Institute of the University of Southern California) on an HP 320 workstation with color graphics. SNePS as well as TINA, the GDK interpreter, are coded in Common LISP. (TINA stands for "TINA Is No Acronym".) The necessary graphics primitives are implemented using primitives of the X window environment.

Natural language parsing is done by using a semantic grammar (Burton & Brown, 1979) based on the ATN formalism (Woods, 1970; Bates, 1978). The used ATN is extended over the version described in Bates (1978) and is part of the standard SNePS distribution (Shapiro, 1982).

The grammar written for this investigation categorizes input into assertions, questions, and commands. Assertions result in building GDK structures that represent the language utterances. Questions activate retrieval operations and return the results of these operations, sometimes combined with a canned phrase. Whenever possible, questions also activate calls to the display function, such that

replies are given graphically as well as with language. Commands always activate calls that change the display.

During natural language input a user might refer to a "form" which is unknown to the system. In this case, the grammar invokes a graphics editor (called Readform) that permits the user to design the referenced form-icon. After exiting the editor, the user will find himself again in the language interaction environment.

Diagram display is performed by the TINA program which is activated from the grammar and understands a number of options, e.g. the number of part levels to be displayed. In Geller (1988) twelve extended test runs demonstrate the abilities of our system and the interactions between language input and graphics output.

4.1. TINA USED AS MAINTENANCE INTERFACE

The use of the TINA program as a graphics interface of the VMES project has been described in a number of earlier publications (Shapiro, Srihari, Taie & Geller, 1986; Taie, Geller, Srihari & Shapiro, 1987; Geller, Taie, Shapiro & Srihari, 1987; Taie, 1987). The VMES system consists of a maintenance reasoner and a graphics interface. The graphics interface is an application of an older version† of the TINA program. The task of the maintenance reasoner is to identify a faulty component in a given device, usually a circuit board. The maintenance reasoner and the display program share a knowledge base realized as a SNePS network.

During the process of identifying a faulty component in a device, the maintenance reasoner repeatedly updates the shared knowledge base. It categorizes components as being in a "default state", being in a state of violated expectation, being recognized faulty or being suspected to be faulty. Information about any of these states is asserted in the network, using the attribute case frame described earlier on. Whenever the maintenance reasoner wants to express changes in its state of knowledge about the analysed device, it executes a call to TINA. TINA presents the current state of the maintenance process to the user. This is done by mapping attributes into signal colors (red = faulty, blue = default, green = suspect, magenta = violated expectation).

Typically a device will be displayed completely blue in the beginning. After finding a violated expectation, for instance a port that has a wrong voltage value, this port will receive an attribute "violated expectation". The device will now be blue, except for the port in question which will be magenta. Finally, after several steps of reasoning and redisplay, the device will be shown in blue with the faulty component(s) in red.

The procedural interface between maintenance reasoner and display program consists of the TINA function only! All other communication is done through the shared knowledge base that both parts of the program have access to. Our experience with this type of programming has been that it is exceedingly easy to combine two independently developed modules. To our own surprise no integratory debugging was necessary!

The most complicated device that was "maintained" with the combined maintenance reasoner/graphics interface has been a six-channel PCM board. In Figure 8 a screen dump from a GIGI terminal is shown. The PCM board does analog/digital

† Based on a VAX 11/780 and a GIGI graphics terminal.

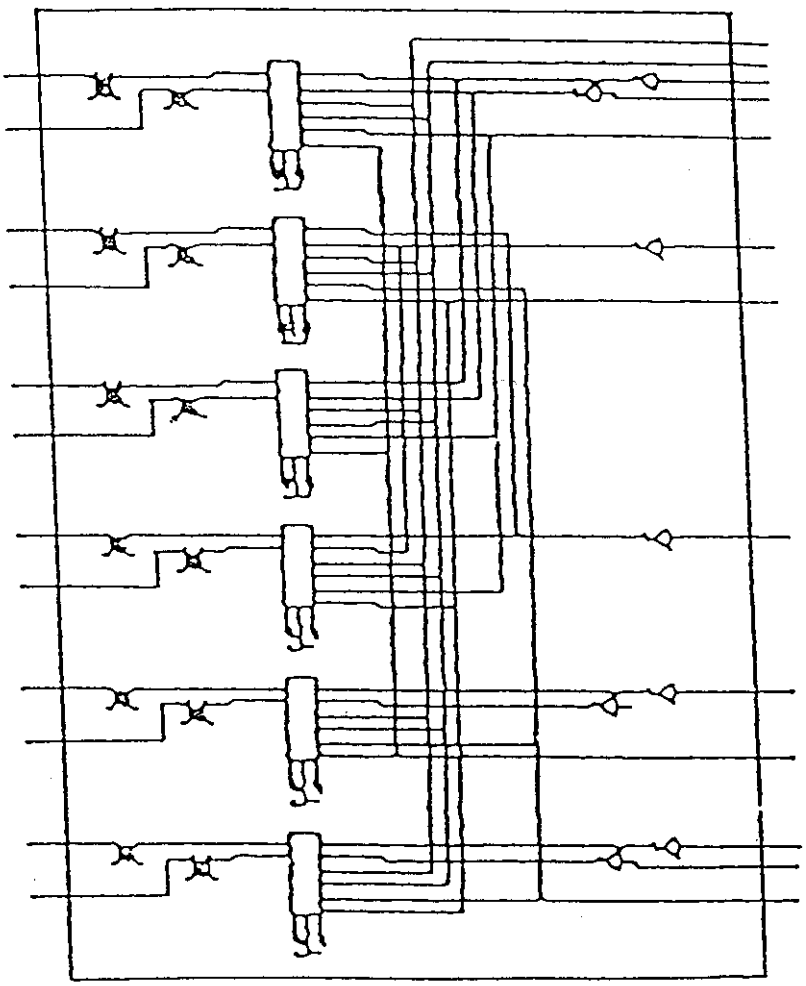

FIGURE 8. A screen dump of the PCM board.

coding and decoding, and its main components are inverters, transformers, and one PCM chip per channel. The large number of components and the limited quality of the involved hardware (printer), unfortunately resulted in a somewhat fuzzy diagram.

5. Intelligent machine drafting

In this section we will briefly present a more specialized application of the GDK theory that has grown out of our work on circuit boards. The problem to be solved is the modeling of the behavior of a draftsman. This problem was introduced by Shapiro and Geller (1986) and also discussed by Geller and Shapiro (1987) and named the "Intelligent Machine Drafting (IMD) Problem".

Before the advent of sophisticated CAD equipment it was the normal way of life

in an engineering company to have developers create (sometimes awful) hand-drawings of the circuits they wanted built. These diagrams then went to the draftsmen who created nicely laid-out wire plans following a few professional conventions. The draftsman does *not* have to understand the functioning of the device he is laying out! It is notable that the job of a draftsman has been considered a low intelligence job, so one should think that AI would have a ready made explanation for "how to do it".

On the other hand over 30 years of AI history have shown that the seemingly easiest problems, such as recognizing a face, are often the most difficult problems, and while highly developed medical advisors, e.g. members of the MYCIN family (Davis, Buchanan & Shortliffe, 1985) have been built, we still have no comprehensive theory of solving many so-called easy problems. It seems to us that it is precisely a "simple" job, such as the job of a draftsman, which requires a lot of *perceptual* intelligence and is therefore difficult for a program to perform.

It has been a part of this project to model the abilities of a draftsman in creating circuit board diagrams. The problem setting considered is slightly different from one a real draftsman is confronted with, because he, as mentioned before, usually bases his work on a hand drawing. In this research the IMD program receives the knowledge base equivalent of a part list, plus complete connectivity information, including the correct inports and outports of every component instead.

At this point the question naturally arises as to whether there is any formal theory of how to draw such circuit board diagrams. A look at textbooks for drafting yields a disappointment (Renton, 1971). One finds only a few conventions and vague explanations. Biesel (1984), whose work concentrates on circuit board diagrams, has collected other evidence for the vagueness of the state of the art in drafting.

IMD is an application of our theory of GDK, because all that is needed for laying out (a class of artificially simple) circuit boards are part, class, form, attribute and inheritability assertions as introduced before. What is omitted from GDK for IMD are position assertions, they are replaced by non-GDK structures that describe ports and connections.

It is important to assert that IMD differs from Computer Aided Design (CAD) in that it deals with functional representations as opposed to structural representations and that the goal of solving a layout and routing problem is to create a "readable" and ideally even "appealing" functional design, as opposed to an optimized structural design.

In Geller (1988) we describe a small class of devices for which we have implemented an IMD module. A report on this work will be published in a later paper.

6. Open problems

Clearly, this work represents only the beginning of a larger effort and does not address a number of important problems. One question to be asked is whether our approach is viable at all. Looking at context effects might raise some doubt about this question.† Graphical representations naturally express context effects, while

† We thank one of our reviewers for pointing out these difficulties.

propositional representations have marked difficulties with them. The work presented here does not attack context problems, but the topic is challenging even without them, and we have decided to work on the aspects of the problem which are accessible to propositional reasoning.

Another seemingly unjustified limitation of our theory is the assumption that there are only two possible views, top and front. What about side views? What about general views? The major focus of our representational theory is two-dimensional diagrammatic representations. Still it captures the most important three-dimensional cognitive distinction, namely the one between a view parallel to the axis of gravitation versus a view orthogonal to the axis of gravitation. Arguments for this major distinction would go beyond the scope of this paper. The treatment of other views and a deeper treatment of three-dimensional phenomena will have to be dealt with in future work.

A similar problem is the limited range of permitted graphical transformations. We are fully aware that a number of interesting phenomena have not been mentioned at all, e.g. shading. Again, we emphasize the limited domain of this work and defer treatment of these problems to a later project.

7. Future work

One direction into which we wish to extend our approach is towards the work of Kosslyn (1981*b*) and to permit "readback" from the generated diagrams to be used for question answering. Specifically we are interested in explaining the examples given by Waltz (1980) that engender surprise in a listener. A preliminary analysis shows that only close interaction between propositional and analog representations can cope with this problem.

Our second main objective is to develop this research towards a knowledge-based user interface management system (KBUIMS). So far it is not possible in our system to define a menu by a combination of natural language and mouse movements, and to link the menu choices to procedures that should be executed on buttoning one of them. However, it is clearly in the range of our paradigm to achieve this effect. The basic goal and unique characteristic of this approach is the complete integration of the language used to interact with the user interface and the language used to interact with the user interface management system.

The current implementation invites improvements in a number of directions. It is not possible to capture dynamic phenomena; for instance one cannot move icons on the screen. The three-dimensional abilities of the program are very limited. The language interface is not general enough, and the program as a whole is not very robust. Pragmatic issues of graphical representations (Geller, 1988; Marks & Reiter, 1990) need to be implemented.

8. Conclusion

The purpose of this paper has been to analyse the knowledge necessary for the aspects of multi-media interfaces that can be captured by the theory of Natural Language Graphics. The notion of Graphical Deep Knowledge has been introduced and used as the leading theme throughout. Graphical Deep Knowledge has been

defined as declarative knowledge that is projectively adequate as well as deductively adequate.

The presented constructs of Graphical Deep Knowledge have been introduced in a frame-like notation for SNePS semantic networks. For many constructs the syntax was given in this case frame format, and a descriptive semantics was applied. In addition procedural effects of introduced structures have been explained informally. The constructs of GDK will now be summarized.

Form descriptions are based on primitive forms (icons) which themselves consist of graphic primitives from a LISP graphics package. They are linked to objects by appropriate case frames, or inherited by objects from classes with an associated form. If no form can be derived this way for an object, exemplar inheritance (a.k.a. up-and-down inheritance) is attempted. We have argued that the latter is a viable technique based on the exemplar view of categorization that should be used if no summary description is available for a class of objects.

A wide range of different position specifications is possible which are all based on one common composed case-frame. Concrete (numerical) as well as fuzzy positions can be represented. Reference objects of position specifications can be given explicitly or deduced from a part hierarchy. Positions can be inherited along the class hierarchy. Different coordinate systems can be defined, and projections explicitly selected. Natural language utterances in reference to objects on the screen can express a view referring to a third dimension. Such utterances are correctly interpreted.

Case frames for relative and absolute attributes have been introduced. Knowledge structures have been introduced that permit the mapping of object attributes into picture attributes. The actual mapping mechanism of an abstract graphics machine is hereby explicated as a function that transforms form-comprising functions into new form-comprising functions with an incorporated attribute.

Part hierarchies, as used in many AI systems, have been replaced by three different part-like hierarchies, namely real parts, assemblies, and clusters. Inheritance of attributes is possible along the part hierarchy of the system. The inheritability of a specific attribute can be expressed declaratively.

SNePS based reasoning in the spatial domain has been discussed. The reasoning facility used in our implementation is called "path-based inference" and permits the description of paths of arcs to derive necessary information about objects. All knowledge retrieval for graphics generation as well as question answering is based on such paths.

In the domain of maintenance systems the use of TINA, our graphics program, was shown as a graphics interface to the VMES (Versatile Maintenance Expert System). A second CAD-like interface based on the same theory was also introduced, which however differs from a CAD system in trying to optimize the "readability" of a pictorial representation. With this interface the completely new field of Intelligent Machine Drafting was defined.

I would like to thank all the many people that have in some way or the other contributed to this work. The list of them in Geller (1988) is almost three pages long. I especially thank Stuart C. Shapiro, my advisor, and Janet Bodner, who corrected my English grammar. Also thanks to my reviewers for pointing out a number of important problems.

References

ABRETT, G. & BURSTEIN, M. H. (1987). The KREME knowledge editing environment. *International Journal of Man-Machine Studies,* **27,** 103–126.

ADORNI, G., DI MANZO, M. & GIUNCHIGLIA, F. (1984*a*). Natural language driven image generation. In *Proceedings of COLING' 84.* Stanford University.

ADORNI, G., DI MANZO, M. & GIUNCHIGLIA, F. (1984*b*). From description to images: what reasoning in between. In *European Conference on Artificial Intelligence,* Pisa.

ARENS, Y., MILLER, L. & SONDHEIMER, N. (1988). Presentation planning using an integrated knowledge base. In *Monterey Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes.*

BATES, M. (1978). The theory and practice of augmented transition network grammars. In L. BOLC, Ed. *Natural Language Communication with Computers. Lecture Notes in Computer Science.* pp. 191–259. New York: Springer Verlag.

BIEDERMAN, I. (1987). Recognition-by-components: a theory of human image understanding. *Psychological Review,* **94,** 115–147.

BIESEL, H. D. (1984). *On Encoding Functional and Schematic Descriptions of Complex Systems.* PhD thesis. Ann Arbor, MI: University Microfilms International.

BORNING, A. (1986). Graphically defining new building blocks in ThingLab. *Human–Computer Interaction,* **2,** 269-295.

BRACHMAN, R. J. (1979). On the epistemological status of semantic networks. In N. V. FINDLER, Ed. *Associative Networks: Representation and Use of Knowledge by Computers,* pp. 3–50. New York: Academic Press.

BRACHMAN, R. J., FIKES, R. E. & LEVESQUE, H. J. (1983). KRYPTON: A functional approach to knowledge representation. *Computer,* **16,** 67–73.

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

BRACHMAN, R. J., FIKES, R. E. & LEVESQUE, H. J. (1985). Krypton: A functional approach to knowledge representation. In R. J. BRACHMAN & H. J. LEVESQUE, Eds. *Readings in Knowledge Representation,* pp. 411–439. Los Altos, CA: Morgan Kaufmann.

BROWN, D. C. & KWASNY, S. C. (1977). *A Natural Language Graphics System.* Technical report OSU-CISRC-TR-77-8, Department of Computer and Information Science, The Ohio State University.

BROWN, D. C. & CHANDRASEKARAN, B. (1981). Design consideration for picture production in a natural language graphics system. *Computer Graphics,* **15,** 174–207.

BURTON, R. R. & BROWN, J. S. (1979). Toward a natural language capability for computer-assisted instruction. In H. O'NEILL, Ed. *Procedures for Instructional System Development,* pp. 273–313. New York: Academic Press.

DAVIS, R., BUCHANAN, B. & SHORTLIFFE, E. (1985). Production rules as a representation for a knowledge-based consultation program. In R. J. BRACHMAN & H. J. LEVESQUE, Eds. *Readings in Knowledge Representation,* pp. 371–387. Los Altos, CA: Morgan Kaufmann.

DI MANZO, M., GIUNCHIGLIA, F. & PINO, E. (1984). Space representation and object positioning in natural language driven image generation. In *International Conference on AI Methodology—Systems—Applications, AIMSA Varna,* Bulgaria.

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

FILLMORE, C. J. (1968). The case for case. In E. BACH & R. T. HARMS, Eds. *Universals in Linguistic Theory,* pp. 1–88. New York: Holt, Rinehart and Winston.

FRIEDELL, M. (1984). Automatic synthesis of graphical object descriptions. *Computer Graphics,* **18,** 53–62.

GELLER, J. & SHAPIRO, S. C. (1987). Graphical deep knowledge for intelligent machine drafting. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* pp. 545–551. Los Altos, CA: Morgan Kaufmann.

GELLER, J., TAIE, M. R., SHAPIRO, S. C. & SRIHARI, S. N. (1987). Device representation and graphics interfaces of VMES. In D. SRIRAM & R. ADEY, Eds. *Knowledge-based*

Expert Systems for Engineering: Classification, Education and Control, pp. 15–28. Southampton, UK: Computational Mechanics Publications.

GELLER, J. (1988). *A Knowledge Representation Theory for Natural Language Graphics.* PhD thesis, published as report #88–15. SUNY at Buffalo, Department of Computer Science.

GUISTINI, R. D., LEVINE, M. D. & MALOWANY, A. S. (1978). Picture generation using semantic nets. *Computer Graphics and Image Processing,* **7,** 1–29.

HAVENS, W. & MACKWORTH, A. (1987). Representing and using knowledge of the visual world. In N. CERCONE & G. MCCALLA Eds. *The Knowledge Frontier,* pp. 429–450. New York: Springer Verlag.

HAYES, P. J. (1977). In defence of logic. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* pp. 559–565. Los Altos, CA: William Kaufman.

HOLLAN, J., MILLER, J., RICH, E. & WILNER, W. (1988). Knowledge Bases and Tools for Building Integrated Multimedia Intelligent Interfaces. In J. W. SULLIVAN & S. W. TYLER, Eds. *Monterey Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes.*

KOSSLYN, S. M. & SHWARTZ, S. P. (1977). A simulation of visual imagery. *Cognitive Science,* **1,** 265–295.

KOSSLYN, S. M. (1980). *Image and Mind.* Cambridge, MA: Harvard University Press.

KOSSLYN, S. M. (1981*a*). Research on mental imagery: some goals and directions. *Cognition,* **10,** 173.

KOSSLYN, S. M. (1981*b*). The medium and the message in mental imagery: a theory. *Psychological Review,* **88,** 46–66.

KOSSLYN, S. M. (1985). Stacking the mental image. *Psychology Today,* May, 23–28.

KUMAR, D. (1989). An integrated model of acting and inference. *Current Trends in SNePS—Semantic Network Processing System,* pp. 55–65. New York: Springer-Verlag.

LEYTON, M. (1986). A theory of information structure II: A theory of perceptual organization. *Journal of Mathematical Psychology,* **30,** 257–305.

LYONS, J. (1977). *Semantics.* New York: Cambridge University Press.

MACGREGOR, R. & BATES, R. (1987). *The LOOM Knowledge Representation Language.* Report ISI/RS-87-188. USC/Information Sciences Institute.

MARKS, J. & REITER, E. (1990). Avoiding unwanted conversational implicature in text and graphics. In *Proceedings of the Eighth National Conference on Artificial Intelligence,* pp. 450–456. Cambridge, MA: The MIT Press.

MCDERMOTT, D. (1981). Artificial intelligence meets natural stupidity. In J. HAUGELAND, Eds. *Mind Design,* pp. 143–160. Cambridge, MA: The MIT Press.

NEAL, J. G. & SHAPIRO, S. C. (1987). Using declarative knowledge for understanding natural language. In L. BOLC, Ed. *Natural Language Parsing Systems,* pp. 49–92. New York: Springer Verlag.

NEAL, J. G. & SHAPIRO, S. C. (1988). Intelligent multi-media interface technology. In J. W. SULLIVAN & S. W. TYLER, Eds. *Monterey Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes.*

NECHES, R. & KACZMAREK, T. (1986). *AAAI-86 Workshop on Intelligence in Interfaces,* Philadelphia.

PAPALASKARIS, M. A. & SCHUBERT, L. (1981). Parts inference: closed and semi-closed partitioning graphs. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* pp. 304–309. Los Altos, CA: Morgan Kaufmann.

POGGIO, A., GARCIA LUNA ACEVES, J. J., CRAIGHILL, E. J., MORAN, D., AGUILAR, L., WORTHINGTON, D. & HIGHT, J. (1985). CCWS: A computer-based, multimedia information system. *Computer,* **18,** 92–102.

RENTON, B. A. (1971). *Electrical and Electronics Drafting.* New York: Hayden Book Company.

REQUICHA, A. A. G. (1980). Representation for rigid solids: theory, methods, and systems. *Computing Surveys,* **12,** 437–464.

REYNOLDS, J. K., POSTEL, J. B., KATZ, A. R., FINN, G. G. & DESCHON, A. L. (1985). The DARPA experimental multi media mail system. *Computer,* **18,** 82–91.

ROBINS, G. (1986). *The NIKL Manual.* Marina Del Rey, CA: University of Southern California, Information Sciences Institute.

ROSCH, E. (1978). Principles of categorization. In E. ROSCH & B. LLOYD, Eds. *Cognition and Categorization,* pp. 27–48. Hillsdale, NJ: Lawrence Erlbaum.

SAMAD, T. (1986). *A Natural Language Interface for Computer-Aided Design.* Boston: Kluwer Academic.

SHAPIRO, S. C. (1979). The SNePS semantic network processing system. In N. V. FINDLER, Eds. *Associative Networks: The Representation and use of Knowledge by Computers,* pp. 179–203. New York: Academic Press.

SHAPIRO, S. C. (1982). Generalized augmented transition network grammars for generation from semantic networks. *The American Journal of Computational Linguistics,* **8,** 12–25.

SHAPIRO, S. C. & THE SNEPS IMPLEMENTATION GROUP, (1983). *SNePS User's Manual.* SNeRG Bibliography #31. SUNY at Buffalo.

SHAPIRO, S. C. & GELLER, J. (1986). Artificial intelligence and automated design. In A. C. HARFMANN, Y. E. KALAY, B. R. MAJKOWSKI & L. M. SWERDLOFF, Eds. *1986 SUNY Buffalo Symposium on CAD, The Computability of Design,* SUNY at Buffalo.

SHAPIRO, S. C., SRIHARI, S. N., TAIE, M. R. & GELLER, J. (1986). VMES: a network based versatile maintenance expert system. In *Proceedings of the First International Conference on Applications of AI to Engineering Problems,* pp. 925–936. New York: Springer Verlag.

SHAPIRO, S. C. & RAPAPORT, W. J. (1986). SNePS considered as a fully intensional propositional semantic network. In *Proceedings of the Fifth National Conference on Artificial Intelligence,* pp. 278–283. Los Altos, CA: Morgan Kaufmann.

SMITH, E. E. & MEDIN, D. L. (1981). *Categories and Concepts.* Cambridge, MA: Harvard University Press.

SMITH, R. G. (1983). STROBE: Support for Structured Object Knowledge Representation. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* pp. 855–858. Los Altos, CA: Morgan Kaufmann.

SONDHEIMER, N. K. (1976). Spatial reference and natural-language machine control. *International Journal of Man–Machine Studies,* **8,** 329–336.

SULLIVAN, J. W. & TYLER, S. W. (1988). Architectures for intelligent interfaces: elements and prototypes. In *Monterey Workshop: ACM/SIGCHI.*

TAIE, M. R. (1987). *Representation of Device Knowledge For Versatile Fault Diagnosis.* PhD thesis, published as report #87-07. SUNY at Buffalo, Department of Computer Science.

TAIE, M. R., GELLER, J., SRIHARI, S. N. & SHAPIRO, S. C. (1987). Knowledge based modeling of circuit boards. In *Proceedings of the Annual Reliability and Maintainability Symposium,* pp. 422–427.

TRANOWSKI, (1988). A knowledge acquisition environment for scene analysis. *International Journal of Man–Machine Studies,* **29,** 197–213.

WALTZ, D. L. (1980). Understanding scene descriptions as event simulations. In *The Proceedings of ACL,* pp. 7–11.

WINSTON, M. E., CHAFFIN, R. & HERRMANN, D. (1987). A taxonomy of part-whole relations. *Cognitive Science,* **11,** 417–444.

WOODS, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM,* **10,** 591–606.

ZDYBEL, F., GREENFELD, N. R., YONKE, M. D. & GIBBONS, J. (1981). An information representation system. In *The Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* pp. 978–984. Los Altos, CA: Morgan Kaufmann.