Chapter 2

Background: Visual Languages and their Domains

This chapter focuses on visual languages; in particular, the field is narrowed (slightly) by concentrating on *concept map*-based visual languages. Here, a concept map consists of nodes and arcs and possibly other constructs, such as contexts. Thus, visual languages based on matrices (spreadsheets), forms (query by example), purely spatial relationships (iconic sentences), and simple text layout (outlining tools) are eliminated. All of the example languages used in this chapter either fall into the category of concept maps, or have concept map languages which implement them. For example, Conceptual Graphs has both a graphical form (concept map) and a "linear form" (textual); and Gaines (1991b) has developed a visual language for CLASSIC, called KDraw as part of his Knowledge Representations System (KRS).

Although the focus of this thesis is on formal and semi-formal visual knowledge representation, informal representations are not completely neglected, for they are often useful (Kremer 1994; 1995):

- in preliminary knowledge gathering where the appropriate formal structure is not apparent,
- when preliminary work is carried out by domain experts who are novices at knowledge representation techniques, or
- when the situation demands a relaxation of formal discipline (such as during an intense meeting)

In addition, Novak and Gowin (1984) describe the use of informal concept maps in education.

2.1 Visual Languages

A visual language, as the term is used here, is any form of communication that relies on two- or three-dimensional graphics rather than simply (relatively) linear text. Even pure text is often supplemented with visual properties such as indentation and even font styles such as bolding and italics. For example, textual programming languages are far more readable if indented to indicate the hierarchical structure; the following code:

```
void makeupper(char* buf) {char* i; for (i=buf; *i; i++) {*i
= toupper(*i);}}
```

is much more difficult to comprehend than the same code with appropriate indentation:

```
void makeupper(char* buf)
      {
      char* i;
      for (i=buf; *i; i++)
            {
            *i = toupper(*i);
            }
      }
```

In spite of the visual aids, this is still considered a textual language. Visual languages all involve pictures of some sort – whether as simple as nodes and connecting arcs, or as complex as nested, multi-dimensional sentences of iconic pictures. Of course, visual languages often involve text, but text's role is usually limited to labels or brief explanations or annotations.

2.1.1 The Value of Visual Languages

In certain circumstances, visual representations have significant advantages over (relatively) linear textual representations. Besides the old adage "a picture is worth a thousand words", Nosek and Roth (1990) have concluded that semantic networks (graphs) are more "computationally efficient" (understandable) than predicate logic (linear text). Two dimensional representations of programs, such as flow charts and even the indentation of textual programming languages, are extremely helpful to aid program understanding (Smith 1977). There are many visual programming languages and languages for program visualization (Myers 1990; Marks 1991; Price, Baecker & Small 1993) as well as large numbers of visual languages for other purposes such as decision making (Conklin & Begeman 1987), knowledge representation (Sowa 1984; Gaines 1991a), geographic information system queries (Calcinelli & Mainguenaud 1994; Aufaure-Portier 1995), and database queries (Constabile & Catarci 1995), to name just a few.

It can be argued that not all people are good at reading graphical representations of data and knowledge. The point seems obvious, but Petre and Green have found evidence that "seeing" the information in a graphical display is an acquired skill (Petre & Green 1993). In contrast to Nosek and Roth's study, Petre and Green's study found that subjects could solve problems faster with their textual languages than with their graphical languages; however, they conclude that although people seem to have to work harder to read graphical information, the graphical notations may prove much richer for expert users. Indeed, there is still not sufficient evidence to evaluate the usefulness of visual languages. Myers (1990) notes the lack of evidence:

> There are not many Visual Languages that would be generally agreed are 'successful', and there is little in the way of formal experiments or informal experience that shows that Visual Languages are good. It would be interesting to see experimental results that demonstrated that visual programming techniques or iconic languages were better than good textual methods for performing the same tasks. Metrics might include learning time, execution speed, retention, etc. Fortunately, preliminary results are appearing for the advantages of using graphics for teaching students how to program. (p. 117)

Regardless of the lack of firm evidence, there is much intuitive appeal to graphical languages. Smith (1977) describes the psychological motivations for using visual languages for programs and data. Among Smith's many arguments are:

- There is evidence to support the notion that abstract reasoning is pictorial in nature.
- The two- and three-dimensional nature of visual reasoning may be more efficient than linear verbal language.
- Visual organizations are an extremely efficient "chunking" method to increase the limit capacity of short-term memory.
- Thought operations may be transformations of images.
- Images are usual analogous to tasks, while linear languages are mostly *Fregean* (bear no analogous relation to the task they describe). Analogical systems are more accurate than Fregean systems.
- Some experiments indicate that the mind is capable of storing sensory data in great detail.

One may also glean some empirical evidence for the power and convenience of visual languages from the experience of Charles Sanders Peirce. Sowa (1984) based his Conceptual Graphs work on Peirce’s much earlier work:

> "The standard notation for symbolic logic was developed by mathematicians like Peano, Russell, and Whitehead, who patterned it after algebra. But one of the pioneers of logic, Charles Sanders Peirce, switched to a graph notation after making major contributions to the linear form. His *existential graphs*, which he called ‘the logic of the future,’ form the logical basis for conceptual graphs." (p. 138)

While Peirce's graphical notations (visual logic languages) have not been well-accepted by mainstream logicians, his invention and use of them argue for their utility for this "pioneer of logic". Peirce himself writes:

> "I do not think I ever *reflect* in words: I employ visual diagrams, firstly because this way of thinking is my natural language of self-communion, and secondly, because I am convinced that it is the best system for the purpose [Ms 620, p. 8]" (Roberts 1973, p. 126)

Roberts (1973), in his book about Pierce's existential graphs, argues for the usefulness of the visual logic language:

> "Logic diagrams these days are certainly less familiar than algebraic notations of logic. But are they really too cumbersome? Would it be fair to say that they are "less perspicuous than the usual" notations, as some say of the Polish notation (Church 1956)? Well, if a facile and perspicuous notation is one that can be quickly learned and easily manipulated, then years of experience with university students have convinced me that EG is the most perspicuous, and *Principia* notation the least. The unusual ease with which inferences can be drawn in EG is something of an unexpected bonus." (p. 126)

The current popularity of direct manipulation interfaces (Shneiderman 1986) is completely compatible with visual languages since many are subject to manipulation with a mouse or other pointing device.

2.1.2 The Meaning of Visual Languages

Like all languages, visual languages can only be reliably interpreted if they have a formal syntax and semantics, though many visual languages lack semantics or both (Myers 1990). Visual languages seem to lag well behind textual languages in this regard. Most formal textual languages (programming languages) are well understood because of the amount of work put into investigating their formal semantics. Little equivalent work has been done for visual languages. This work is described in this section.

Crimi et al (1991) describe the application of relational grammars to multi-dimensional (visual) languages in order to define a visual syntax. In their formulation, a context-free relational grammar is a tuple,

$$G = (V_N, V_T, V_R, S, P, R)$$

where:

- V_T is a finite set of terminal symbols.
- V_N is a finite set of non-terminal symbols.
- V_R is a finite set of relation symbols, $V_R = \{r_i | 1 \leq i \leq k\}$; for each r_i its arity $a(r_i)$ is defined.
- S is the starting symbol.
- P is a finite set of productions, where each production has the form A::=M,R where: $A \in V_N$; M is a multiset of symbols in $V_N \cup V_T$; and R is set of *constraints* over the symbols in M which must hold for a production to be applied.
- R is a set of *evaluation rules* used to validate constraints.

The constraints, R, in the productions, P, are defined by the evaluation rules, R. For example, the set of all possible untyped directed graphs is defined by (V_N, V_T, V_R, G, P, R), where ("O" represents a node and "→" represents a directed arc):

```
VN = {G}
VT = {O,→}
VR = {start,end}
P =   1. G::={O}
      2. G::={O,G}
      3. G::={→,G} {start(→,G), end(→,G)}
R =   1. start(→,G) :- (G ⇒ {O}), start(→,O)
      2. start(→,G1) :- (G1 ⇒ {O,G2}), start(→,O)
      3. start(→,G1) :- (G1 ⇒ {O,G2}), start(→,G2)
      4. start(→1,G1) :- (G1 ⇒ {→2,G2}), start(→1,G2)
      5. end(→,G) :- (G ⇒ {O}), end(→,O)
      6. end(→,G1) :- (G1 ⇒ {O,G2}), end(→,O)
      7. end(→,G1) :- (G1 ⇒ {O,G2}), end(→,G2)
      8. end(→1,G1) :- (G1 ⇒ {→2,G2}), end(→1,G2)
```

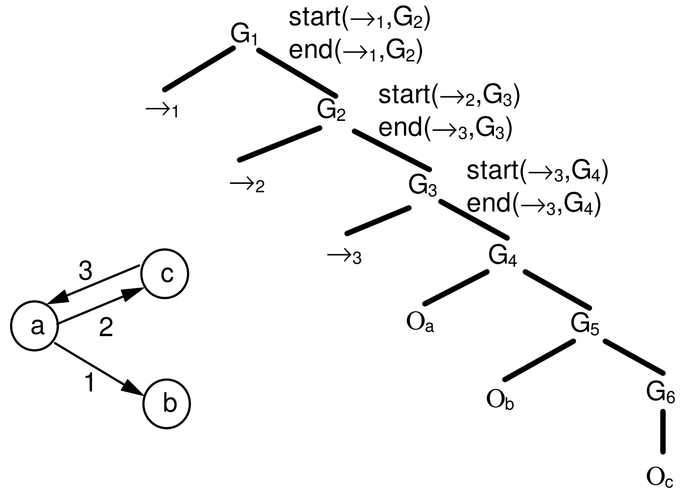

Figure 4: A directed graph and its parse tree (Crimi et al. 1991)

Production 1 says that a single node is a legal graph. Production 2 says a single node can always be added to a graph. Production 3 says a directed arc can only be added to a graph if it starts and ends in the graph.

The evaluation rules, R, specify how the start and end relations are to be interpreted. They are defined recursively in a Prolog-like language. For example, Rule 1 states that a directed arc starts on G if G matches a single node ($G \Rightarrow \{O\}$), and the arc starts on that node (start(→,O)). Rules 2 and 3 state that a directed arc starts on G_1 if G_1 matches $\{O,G_2\}$ and the arc starts on the node (start(→,O)) or on a node in G_2 (start(→,G_2)). Figure 4 shows a simple directed graph and its parse tree according to this relational grammar.

In contrast, Wang and Lee (1993) do not concern themselves with syntax, but go a step in the direction of semantics. They describe a formal semantics for visual reasoning. In their formulation, graphs are given meaning by an *interpretation* that maps the graphical domain to an application domain:

$$\boxed{\text{graphical domain}} \xrightarrow{\text{interpretation}} \boxed{\text{application domain}}$$

A *graphical signature*, Σ, is a quadruple ($\mathcal{T}$, $\leq$, $\mathcal{F}$, $\mathcal{P}$) where:

- ($\mathcal{T}$, $\leq$) is a partially ordered set of sorts
- $\mathcal{F}$ is a set of function symbols of the form $s_1 \times s_2 \times \ldots \times s_n \rightarrow s$, where $s_i, s \in \mathcal{T}$
- $\mathcal{P}$ is a set of predicate symbols of the form $s_1 \times s_2 \times \ldots \times s_n$, where $s_i \in \mathcal{T}$

Thus, $\mathcal{T}$ is the set of graphical object types in the graphical domain (e.g.: circle $\leq$ closed_curve) , and $\mathcal{F}$ and $\mathcal{P}$ are just the set of functions (e.g.: overlap: circle $\times$ circle $\rightarrow$ closed_curve) and predicates (e.g.: overlapping: circle $\times$ circle) defined on them. A *domain language* has a similar signature, $\Sigma_{\mathcal{D}}$, and an interpretation, $\mathcal{I}$: $\Sigma \rightarrow \Sigma_{\mathcal{D}}$, defines functions from the element of $\mathcal{T}$, $\mathcal{F}$, and $\mathcal{P}$ in Σ to the elements of $\mathcal{T}_{\mathcal{D}}$, $\mathcal{F}_{\mathcal{D}}$, and $\mathcal{P}_{\mathcal{D}}$ in $\Sigma_{\mathcal{D}}$ respectively (where each of the functions must preserve the partial order, type, and domain, respectively). Table 1 gives an example interpretation from a graphical domain of circles and points to an application domain of sets and elements.

2.1.3 Classification of Visual Languages

There are several taxonomies of visual languages, though most focus on specific domains. For example, Price, Baecker, and Small (1993) focus only on programming visualization languages. Myers (1990) concentrates on the area of *visual programming* and *program visualization* and describes three separate taxonomies. Visual programming is the specification of programs using graphical techniques while programming visualization is using graphical techniques to explicate the structure of a (not necessarily visual) program. Myers' first taxonomy is about programming systems and is based on the following three dimensions:

- visual programming vs. not visual programming: differentiates true visual languages from traditional text-based languages.

- example-based programming vs. not example-based programming: differentiates systems that attempt to infer programs from one or more examples from those that require explicit specification.
- interpretive vs. compiled: differentiates systems that can be directly executed from those that require an intermediate translation before they can be executed.

Myers' second taxonomy is of particular interest to this work and classifies systems into 14 categories of *specification technique*. Table 2 lists Myers' specification techniques showing the six types that are kinds of concept maps. It is interesting to note that not only are six of Myers' 14 classifications different forms of concept maps, but that fully 25 of his 39 non-textual-language examples fall into these six concept map categories.

Table 1: An interpretation from the graphical domain to the domain of set theory

Σ	$\mathcal{T}$	Point, Circle
	$\leq$	$s \leq s$ for all $s \in \mathcal{T}$
	$\mathcal{F}$	a,b:Point, A,B:Circle
	$\mathcal{P}$	in_1:Point $\times$ Circle; in_2:Circle $\times$ Circle
Σ_D	$\mathcal{T}_D$	Element, Set
	$\leq_D$	…
	$\mathcal{F}_D$	a,b:Element, A,B:Set
	$\mathcal{P}_D$	$\in$:Element $\times$ Set; $\subseteq$:Set $\times$ Set
$\mathcal{I}$	$\mathcal{I}_T$	Point $\rightarrow$ Element, Circle $\rightarrow$ Set
	$\mathcal{I}_F$	A $\rightarrow$ A, B $\rightarrow$ B, a $\rightarrow$ a, b $\rightarrow$ b
	$\mathcal{I}_P$	$in_1 \rightarrow \in$, $in_2 \rightarrow \subseteq$

Myers' third taxonomy divides program visualization systems according to whether they represent code, data, or algorithms.

There are other taxonomies of visual languages, but Myers' are the most well-known, and the others tend to specialize within a specific area. For example, Price, Baecker and Small quantitatively grade their 12 example program visualization languages in each of over 30 leaf-node categories arranged in a four level hierarchy (Price, Baecker & Small 1993).

Table 2: Myers' taxonomy of specification techniques (Myers 1990)

	Textual languages
Concept maps	Flowcharts
	Flowchart derivatives
	Petri nets
	Data flow graphs
	Directed graphs
	Graph derivatives
	Matrices
	Jigsaw puzzle pieces
	Forms
	Iconic sentences
	Spreadsheets
	Demonstrational
	None

2.2 Concept Maps

Concept map is a very general term that describes a simple and intuitive type of visual language. For human users, concept maps tend to make the structure of a body of knowledge much more salient than other forms of knowledge representation such as pure text and predicate logic (Nosek & Roth 1990). Concept maps have been used in education (Lambiotte et al. 1984; Novak & Gowin 1984), in management (Axelrod 1976; Hart 1977; Eden, Jones & Sims 1979; Banathy 1991), in artificial intelligence (Quillian 1968), in knowledge acquisition (McNeese et al. 1990), in linguistics (Sowa 1984; Graesser & Clark 1985) and for many other varied purposes. Concept maps can (and have) been used to represent knowledge at the very informal level, such as for "brainstorming", as well as at the very formal level, such as executable expert systems (Gaines 1991b) or graphic forms (Ellis & Levinson 1992; Eklund, Leane & Nowak 1993) of Conceptual Graphs (Sowa 1984) (see also (Kremer 1994; 1995)).

Concept maps are graphs (in the mathematical sense of the word) consisting of nodes with connecting arcs, which represent relationships between nodes (Lambiotte et al. 1984). The nodes are labeled with descriptive text, representing the “concept”, and the arcs are often labeled with a relationship type. Nodes may be represented using distinct visual attributes, such as shape and color, to distinguish node types; arcs may be similarly distinguished. Thus, concept maps model a simple graph theory. The notion of concept map may be extended to include *contexts*, boxes that contain a subgraph (sub-concept map) of the graph (concept map), which models *partitions* in graph theory.

Figure 5 is an example of an informal concept map developed by a group during an interactive brainstorming session about a new network in a large company. In contrast, Figure 6 (page 35) shows a formal concept map, called a Conceptual Graph (Sowa 1984), which models the sentence "Tom believes Mary wants to marry a sailor" in a way that is amenable to straightforward, unambiguous interpretation by a computer program.

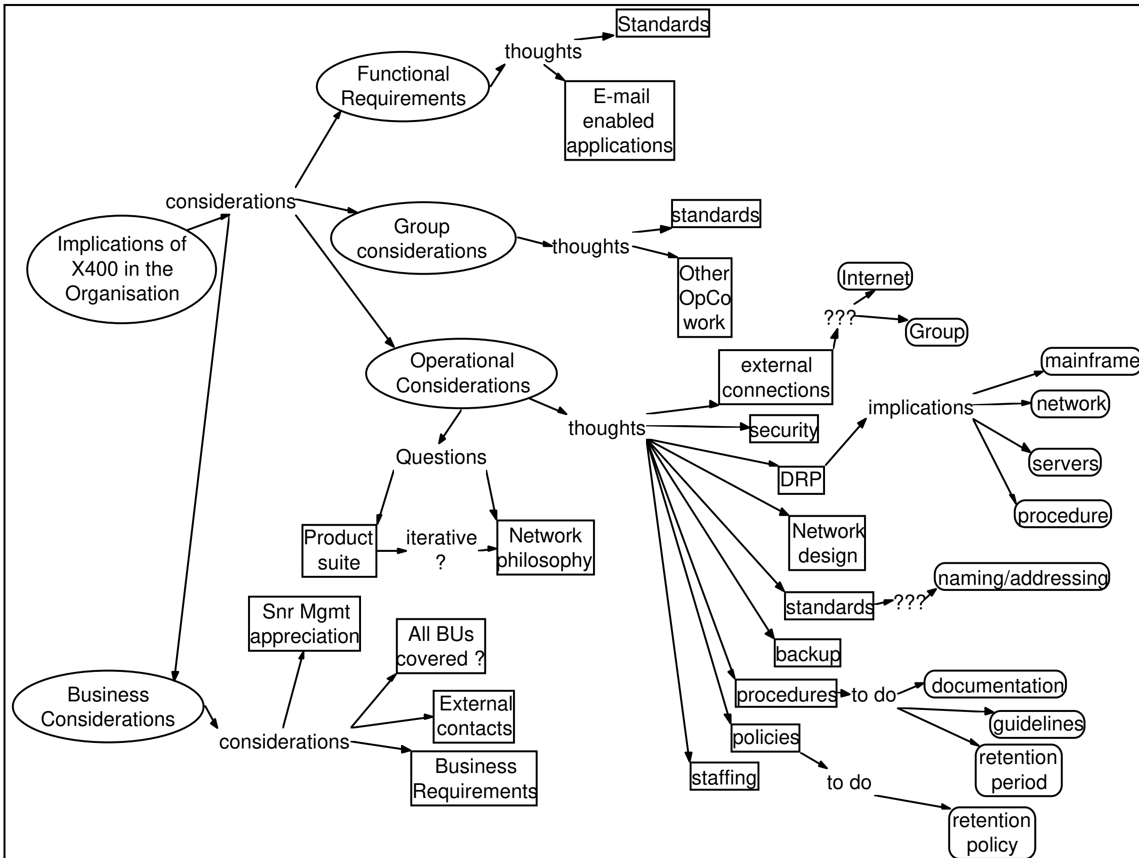

Figure 5: A concept map developed by a three member group brainstorming network design (Kremer 1993).

Table 2 (page 27) shows where concept maps fit into Myers' taxonomy of visual languages, subsuming six of his 14 categories and constituting 24 of his 39 example visual languages. Any of the various forms of concept maps may be described in terms of Crimi et al's relational grammars (Crimi et al. 1991) in a manner very similar to the example given in Section 2.1.2. This example, however, only considers directed graphs, while concept maps may contain directed, undirected, and bi-directional arcs – or a mix of these. The example is also overly simple in that it assumes only untyped nodes and arcs, while concept mapping languages usually include typed nodes and/or typed arcs where the rules include constraints on the interconnections based on type.

In terms of Wang and Lee's semantics (Wang & Lee 1993), the term *concept map* is too general to be assigned a mapping to a specific application domain: there are many concept mapping languages which each map to their own set of application domains. However, one can describe the graphical signature, Σ, of concept maps at a high level (*top* and *null* are the universal type and the absurd type, respectively):

Σ: $\mathcal{T}$ top, pos_int, graph, node, arc, null

$\leq$ graph $\leq$ top, node $\leq$ graph, arc $\leq$ graph, null $\leq$ node, null $\leq$ arc, pos_int $\leq$ top, null $\leq$ pos_int

$\mathcal{F}$ anchored_to: arc $\times$ pos_int $\rightarrow$ node

$\mathcal{P}$ anchored: arc $\times$ pos_int $\times$ node

This description specifies that graphs contain nodes and arcs, but no common subtype between the two. It specifies that arcs may be connected only to nodes, but this may not necessarily be a restriction in all concept mapping languages (arcs may sometimes connect to other arcs). Furthermore, Σ is quite general in that it allows for arcs of arbitrary arity, where most (but not all) concept mapping languages restrict themselves to binary arcs. The function *anchored_to* could be used to determine the node at the *i*th terminal of an arc; the predicate *anchored* can be used to test whether the *i*th terminal of an arc is attached to a certain node. If this description were to be extended to include contexts, a (non-universal) common supertype of *node* and *arc* would have to be introduced to allow both to be included in partitions.

Lambiottte, Dansereau, Cross, and Reynolds (1984) classify “multirelational semantic maps” into four categories:

- Spatial layout and location systems: These systems rely on spatial proximity, spatial relationship (e.g.: above or below), or boundary lines to convey relationships among items. Layout systems are limited in complex domains because it is difficult to convey more than a very few relationship types spatially.

- Node based systems: These systems generally use node attributes (e.g.: shape, color, or style) to distinguish a variety of nodes types to convey information. If arcs are used at all, the relationship they represent is ambiguous, or must be inferred from the adjoining node types.
- Link-based systems: These systems convey almost all information in arcs which may be labeled and/or otherwise distinguished by attributes such as style, thickness or color to convey their type either canonically (drawn from a set of predefined types) or idiosyncratically (ad-hoc types). If nodes are present at all, they contain only the data, and are uniform in appearance.
- Hybrid systems: These systems take advantage of spatial layout, node type, and link type to convey a much richer set of semantic information.

The first category, spatial layout, may not be considered a concept map at all. The other three categories constitute concept maps with typed nodes but untyped arcs, concept maps with untyped nodes but typed arcs, and concept maps with both typed nodes and typed arcs. Although spatial layout does not usually effect the explicit meaning in concept mapping languages, it does have an effect on the understandability of a particular map. Concept mapping languages with context do use spatial layout in that containment in contexts usually conveys semantic meaning.

The remainder of this chapter focuses on examples of concept mapping languages. These include two languages for knowledge representation and one language for decision making.

2.3 Visual Knowledge Representation Languages

Almost any language (including English) can be conceived of as a knowledge representation language. But the emphasis here is on formal and semi-formal concept mapping languages with (almost) unambiguous semantics. That is, this thesis does not focus on natural languages (like English), but instead concentrates on "artificial" languages like CLASSIC (Borgida et al. 1989) and Conceptual Graphs (Sowa 1984) whose primary intent is to render knowledge in a form amenable to computational support. CLASSIC and its visual counterpart, KDraw (Gaines 1991b), seek to capture knowledge such that inferences can be made to classify the domain elements. One of Conceptual Graphs' goals is to represent natural language expressions in a graphical form, but sufficiently disambiguated so as to render the representations amenable to computer processing. These are "pure" knowledge representation languages. Also of interest are languages with less lofty goals – some of these are discussed in subsequent sections.

In order to illustrate visual knowledge representation languages, this section describes two very different visual knowledge representation languages: Gaines' KDraw, a visual form of CLASSIC; and Sowa's Conceptual Graphs, which has both a graphical and linear (textual) form. Coincidentally, these two languages are both used as examples in Chapter 7.

2.3.1 KDraw

CLASSIC (Borgida et al. 1989) is a textual knowledge representation language in the KL-ONE family of languages. CLASSIC is an "open world" database, as opposed to the relational data base model which embraces a "closed world" assumption (if it is not in the database it doesn't exist). Users can populate the CLASSIC database with *concepts*, *primitives,* and *individuals* and assign *roles* (like fields in relational databases, or slots in object oriented systems) to the concepts, primitives, and individuals. The roles can be constrained as to type (concept conformance) and arity, and can be assigned values in the form of individuals. For example, a user can add the concept CRIME to a CLASSIC database as follows:

```
define-role[perpetrator]
define-role[victim]
…
define-concept[CRIME,
      (PRIMITIVE
            (AND  (AT-LEAST 1 perpetrator)
                  (ALL perpetrator PERSON)
                  (AT-LEAST 1 victim)
                  (AT-LEAST 1 site)
                  (AT-MOST 1 site))
      crime)].
```

CLASSIC works by classifying *individuals* as members of one or more *concepts* in a concept hierarchy. Information about concepts and individuals is encoded as *roles* (attributes or slots) and associated *values*. CLASSIC matches the roles and values of individuals to infer which concepts an individual belongs to. Once an individual has been classified, *rules* associated with the concepts may be fired to further instantiate knowledge about the individual (by filling in roles and role values).

There has been extensive work on complexity of the subsumption algorithms in KL-ONE languages (Donini et al. 1995). The subsumption algorithm is important because subsumption calculation is the basis of all computation in KL-ONE languages. It turns out that extension of a base level version of KL-ONE (composed of conjunction, limited negation, universal quantification, limited role existential quantification, and numeric role filler limits) leads to worst-case complexities of NP-hard or worse. Fortunately, subsumption seems to be computationally tractable in practical situations.

Gaines has developed a visual version of the CLASSIC language, called KDraw in which the user can specify and run a complete expert system (Gaines 1991b). A small excerpt from a knowledge base about birds is illustrated in Figure 2 on page 4. The ellipse with tick marks identifies *bird* as a primitive concept. The unmarked ellipses labeled *tubnose*, *albatross*, *fulmar*, *layson albatross*, and *black-footed albatross* represent subconcepts of *bird* as shown by the directed arcs. The labels with no surround are roles and the labels with rectangular surrounds are *individuals* which, in this case, are acting as values for the roles. The interpretation is that an observed *individual* of type *bird* may be recognized as a *tubenose* if it is observed that its *nostrils* are *external-tubular*, it *lives at sea*, and its *bill* is *hooked*. The class of the *bird* may be further refined by further observation as pictured. If a *bird* is classed as the concept *black-footed albatross*, then a rule is fired (*rule1* in the barred rectangle) which assigns "Black-footed Albatross" to its *name* role. A complete KDraw knowledge base would have information about many other birds, descriptions of the types of things that can play the value part for each role, and various other pieces of information such as the mutual exclusion among certain concepts.

2.3.2 Conceptual Graphs

Another form of knowledge representation is Conceptual Graphs (Sowa 1984). Conceptual graphs were developed by Sowa based on Peirce's work (see page 21) on existential graphs between 1897 and 1906. Conceptual Graphs were *originally* developed as graphs, although a text format (called "linear form") has evolved for ease of use by computer programs. Computerized graphical editors now exist for Conceptual Graphs, for example the *GrIT* (Ellis & Levinson 1992) system which provides a graphical interface to the PEIRCE system, which is a Conceptual Graphs database and inference engine.

Conceptual graphs are capable of cleanly representing an extremely wide range of knowledge forms – from predicate logic to natural language. For example, Figure 6 is a conceptual graph representation of the surprisingly complex sentence "Tom believes that Mary wants to marry a sailor." Conceptual graphs are bipartite, with squared boxes representing concepts (and individuals) and rounded boxes representing relations between the concepts (and individuals). The larger boxes in Figure 6 are also concepts but are a special kind of concept called contexts. The outer concepts may be read as "There is an

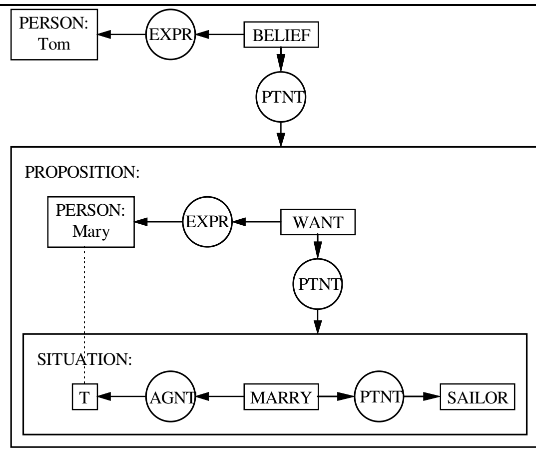

Figure 6: A concept map in the form of Sowa's Conceptual Graphs describing the sentence "Tom believes that Mary wants to marry a sailor." (Sowa 1984)

instance of a belief whose experiencer is the person Tom, and whose patient is a certain proposition." The proposition is "There is an instance of a want whose experiencer is a person Mary and whose patient is a certain situation." The situation is "There is an instance of marriage whose agent is something (T) and whose patient is some sailor." There is also a coreference link (the dotted line) between the concept [Person: Mary] and the concept [T]. This indicates that [Person: Mary] and [T] are the same individual, which modifies the interpretation of the situation to be "There is an instance of marriage whose agent is a person Mary and whose patient is some sailor."

The linear form of the conceptual graph in Figure 6 is

```
[BELIEF] -
 (EXPR) -> [PERSON:Tom]
 (PTNT) -> [PROPOSITION:
             [WANT] -
               (EXPR) -> [PERSON:Mary]
               (PTNT) -> [SITUATION:
                           [MARRY] -
                             (AGNT) -> [T:Mary]
                             (PTNT) -> [SAILOR]
                         ]
           ]
```

which represents the boxes (concepts) in Figure 6 with square brackets and the circles (relations) with round parenthesis. The arcs have fairly obvious translations using "->" and "<-". The one element that does not translate in a straightforward "graphical" manner is the coreference arc (dotted arc); coreference is represented only implicitly by the coincidence of the variable name *Mary*.

The above example serves to show that Conceptual Graphs can represent natural language sentences, but Sowa (1984) shows how Conceptual Graphs can be used to interpret natural language and to generate natural language as well. Furthermore, Conceptual Graphs are shown to represent first order logic (with an elegant set of visual deduction rules) and model logic.

The description of Conceptual Graphs given so far involves only the graphs themselves. A complete Conceptual Graphs definition is a *canon* consisting of the graphs, a type lattice (semantic net), the conformity relations (instance-of relations), and the formation rules. The graphs have already been discussed. The type lattice is a semantic net describing the subtype/supertype relationships among the concept and relation types used in the graphs. The conformity relations list describe the types of all individuals used in the graphs, so are ordered pairs from individuals to the elements in the type lattice. Finally, the formation rules are:

- *Restrict*: specializes the type of concept
- *Copy*: duplicates a graph.
- *Simplify*: removes any redundant (identical) relations between two concepts.
- *Join*: merges two graphs by unifying an identical concept node from each graph into a single concept.

Other operators (such as *maximal join* – joining two graphs on as many concepts as possible) are implemented in terms of these fundamental rules.

There now exist several tools for the storage and manipulation of Conceptual Graphs. These include PEIRCE (Ellis & Levinson 1992), Deakin ToolSet (Lukose 1991), and CGKEE (Munday et al. 1996). These tools are all based on the Conceptual Graphs linear form.

In addition, there have been several extensions to Conceptual Graphs. Lukose has developed Model-ECS (Lukose 1993; Lukose 1995) which is an executable form of Conceptual Graphs.

2.4 Other Concept Mapping Languages

Although visual knowledge representation languages are of primary interest in this thesis, other types of concept mapping languages are of interest as well. For example, gIBIS (Conklin & Begeman 1987) is a simple semi-formal language used in decision making and represents issues, positions, and arguments. There are also a vast number of visual programming languages which are, by definition, quite formal. These include OPAL (Musem, Fagen & Shortliffe 1987) and Pygmalion (Smith 1977) and many others (Myers 1990). Closely related to visual programming languages are program visualization languages (Price, Baecker & Small 1993) and languages used to design or represent programs. These include traditional flowcharting languages, and the newer object oriented programming notations such as Booch Notation (Booch 1994), Rumbaugh Notation/OMT Notation (Rumbaugh et al. 1991), Coad Notation (Coad, North & Mayfield 1995), and the more recent "Unified Method" (Booch & Rumbaugh 1995).

2.4.1 gIBIS

Figure 7 shows a concept map that may be developed by the participants in a decision-making meeting. The visual language used in Figure 7 is a semi-formalism called gIBIS (Graphical Issue Based Information System) (Conklin & Begeman 1987), which is an easy-to-use semi-formal language used for business decision making. Here, the elliptical

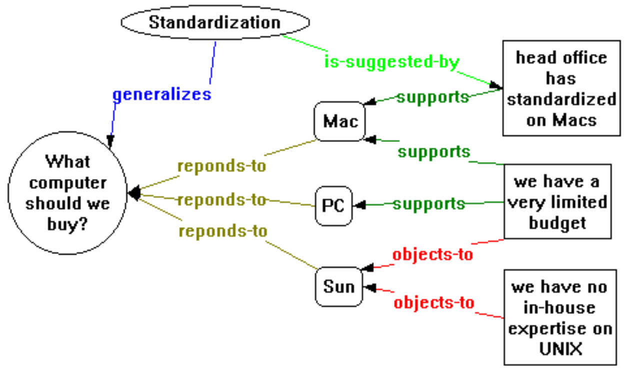

Figure 7: A concept map used to support a decision about buying a computer

nodes represent *issues*, the rounded-rectangular nodes represent *positions* taken on the issues, and the rectangular nodes represent *arguments*. The arcs connecting the nodes are all labeled with relationship types.

The decision making process represented in Figure 7 shows a clear and clean structure that would be quite difficult to represent using plain text. The relative linearity of text would obscure the relationships among the individual arguments and the sub-set of positions they relate to. To be sure, a two-dimensional table could clearly represent the structure of most of Figure 7 (with the exception of the "standardization" part), but tables quickly become difficult as the complexity of the situation increases. The flexible, two-dimensional structure of the concept map is much more capable of clearly representing complex knowledge structures.

2.5 Summary

This chapter has described visual languages as distinct from natural languages (such as English) and other textual languages (such as traditional programming languages (like C or Pascal). Visual languages use pictures to convey meaning in a two- or three-dimensional space. Visual languages range from very informal and unstructured to very formal and constrained. The value of visual languages is evidenced by the wide variety and large number of visual languages in existence, as well as by a number of compelling psychological arguments and some empirical evidence.

The syntax and semantics of visual languages is as varied as visual languages themselves, but Crimi et al (1991) have applied relational grammars to visual languages to formalize their syntax. Furthermore, Wang and Lee (1993) have addressed the semantics of visual languages by formalizing a system to describe an *interpretation* from the graphical (visual) domain to an application domain.

There exist several taxonomies of visual languages; however, Myers (1990) is the most well known and most inclusive. Myers uses three separate taxonomies to classify languages as programming systems, by specification technique, and by what artifacts they represent.

Concept maps are a general kind of visual language and are used for many purposes. Concept maps are graphs with nodes, arcs that interconnect the nodes, and possibly contexts which partition the map. Concept maps may be described in terms of Crimi et al's relational grammar and in terms of Wang and Lee's semantic interpretations, but *concept map* is too general a term to describe specific interpretations: interpretations must be applied to specific applications of concept maps.

Three specific concept mapping languages are described. Gaines' KDraw is an visual version of the CLASSIC knowledge representation language used in artificial intelligence applications. Sowa's Conceptual Graphs is also a knowledge representation language, and is used in natural language interpretation, natural language generation, and reasoning. Finally, a very simple decision-support language, gIBIS, is described.