

An Interactive Visual Language for Term Subsumption Languages

*Brian R. Gaines
Knowledge Science Institute
University of Calgary
Alberta, Canada T2N 1N4
gaines@cpsc.ucalgary.ca*

Abstract

A visual language is defined equivalent in expressive power to term subsumption languages expressed in textual form. To each knowledge representation primitive there corresponds a visual form expressing it concisely and completely. The visual language and textual languages are intertranslatable. Expressions in the language are graphs of labeled nodes and directed or undirected arcs. The nodes are labeled textually or iconically and their types are denoted by six different outlines. Computer-readable expressions in the language may be created through a structure editor that ensures that syntactic constraints are obeyed. The editor exports knowledge structures to a knowledge representation server computing subsumption and recognition, and maintaining a hybrid knowledge base of concept definitions and individual assertions. The server can respond to queries graphically displaying the results in the visual language in editable form. Knowledge structures can be entered directly in the editor or imported from knowledge acquisition tools such as those supporting repertory grid elicitation and empirical induction. Knowledge structures can be exported to a range of knowledge-based systems.

1 Introduction

Visual presentation of knowledge structures has been an attractive feature of semantic networks since their inception. In knowledge acquisition, in particular, the presentation of formal knowledge to those from whom it has been elicited is important to its validation. There are many techniques for such acquisition but they ultimately result in an operational knowledge base with formal semantics. However, the expression of this knowledge base in the formal language used by the system is usually not very comprehensible to non-programmers [Nosek & Roth, 1990]. A visual language that is both comprehensible and formal offers attractive possibilities not only for comprehension but also for editing, and for parts of the knowledge acquisition process itself.

The early development of semantic nets resulted in criticisms that the semantics of particular diagrams was not well-defined [Woods, 1975; Brachman, 1977]. Nodes, arcs and their labels could be used very freely and ambiguously and diagrams were subject to differing interpretations. In the 1970s there were proposals for network formalisms with well-defined semantics [Cercone & Schubert, 1975; Fahlman, 1979; Brachman, 1979]. However, these preceded two important developments in computing: first, the ubiquity of personal workstations with high resolution graphics supporting visual languages as operational editors [Glinert, 1990]; second, the studies of complexity issues in knowledge representation leading from the complexity of KL-ONE [Brachman & Schmolze, 1985] through the logical universality of KRYPTON [Brachman, Gilbert & Levesque, 1985] to the simplified and tractable semantics of CLASSIC [Borgida *et al*, 1989].

In the light of these developments it is timely to re-examine semantic networks as formal visual languages from two perspectives:

- *Cognitive Ergonomics*: is it possible to create visual languages that are simple and natural to use in operational form as interactive structure editors?
- *Formal Semantics*: is it possible to intertranslate (that is, translate unambiguously in both directions) between these languages and knowledge representation formalisms with well-defined semantics?

Note that the second question is deliberately phrased to avoid mixing the issues of knowledge representation, logic, intensional and extensional semantics, and so on, that cast doubt upon the utility of early semantics networks, with the issues of visual representation. It leaves the deep semantic issues in the province of formal knowledge representation where they belong, and assesses the semantic validity of a visual language by its intertranslatability with established formalisms [Mackinlay & Genesereth, 1984]. This is not to say that the requirements for visualization might not in themselves lead to insights relating to the formalisms and semantics—diagrams and notation play an important role in scientific thinking—but this is not the primary criterion for judging a visual knowledge representation language.

Computer production of visual forms of knowledge represented in a computer has been a topic of research since the early days of knowledge representation research [Schmolze, 1983] and a feature of many research systems [Kindermann & Quantz, 1989] and commercial products. Some expert system shells and knowledge acquisition tools have used the power of modern workstations to provide continuously updated graphs of some aspects of a knowledge structure being entered textually. Abrett and Burstein's [1988] KREME system graphically displays the computed subsumption relations between concepts so that those entering knowledge structures can see the consequences of definitions and detect errors due to incorrect or inadequate definitions. However, KREME does not support graphic knowledge entry or editing.

This paper presents the visual syntax and underlying semantics of a visual language for term subsumption languages. It focuses on the use of the language to enter and edit knowledge visually, and on its application in a highly interactive graphic structure editor. However, the language is also well-suited to the display of knowledge structures, and the system includes a grapher using Watanabe's [1989] heuristics for the layout of complex graphs.

2 A Formal Visual Language

The approach to language definition taken in this section is to define each construct in the visual language in terms of its visual appearance, basic semantics, and intertranslation with CLASSIC expressions [Borgida *et al*, 1989].

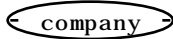
The visual language provides the means to represent knowledge structures as graphs of labelled nodes and arcs. The visual primitives of the language are:

- *Nodes*, identified and typed as specified below.
- Two *arc* types linking nodes—a directed arc and an undirected arc—for this paper they are taken to be a line with, and without, an arrow.


- Text *labels* for the nodes—with an associated equivalence relation, which may take into account case, embedded spaces, and so on, but otherwise based on lexical identity.
- Six distinctive text *surrounds* defining the node types—again subject to choice—for this paper they are taken to be ovals (concepts), marked ovals (primitives), rectangles (individuals), no surround (roles or annotation), rounded corner rectangles (rules), and marked rounded corner rectangles (constraints, e.g. cardinality and set inclusion).

The node labels for concepts and primitives form one equivalence class and those for individuals, roles and rules each form additional separate equivalence classes so that the same string may be used to label nodes of different types without ambiguity. The text in the constraint nodes is restricted to be an allowable constraint, typically on cardinality, set membership or numeric range, but the text in the other five node types is unconstrained. The implementation allows an icons to be made equivalent to a text label and substituted for it visually, but the knowledge representation semantics are unaffected by this substitution.

2.1 Primitive Concepts

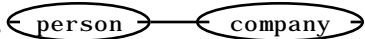
Primitive concepts, and the relations between them, are the foundations for knowledge representation schema. A primitive concept is represented in the visual language as its text label in an oval with marks at either end. For example,  signifies the primitive concept “company”—something may be asserted to be a “company” but cannot be recognized to be one because no sufficient definition has been given. This translates to and from CLASSIC terminology as:

define-concept[company, (PRIMITIVE company)].

In the visual language, the basic relation of *subsumption* between concepts is represented by an arrow. For example,  signifies that “company” is subsumed by “legal entity”—asserting something to be a “company” commits us to asserting that it is also a “legal entity”, the strict ‘is-a’ relation. The concept “company” encodes all the properties encoded by “legal-entity” together with, possibly, others. This translates to and from CLASSIC terminology as:

define-concept[legal-entity, (PRIMITIVE legal-entity)],

define-concept[company, (PRIMITIVE (AND legal-entity) company)].

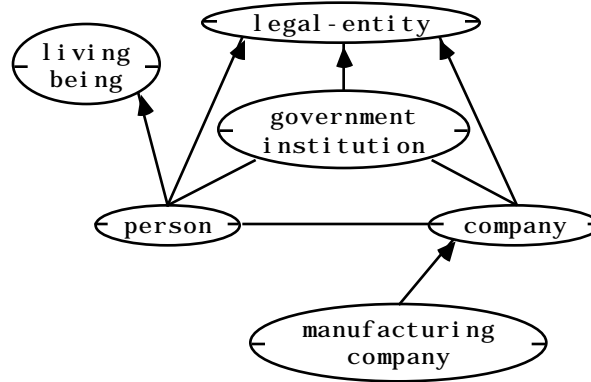
The basic relation of *disjointness* between concepts is represented by an undirected line between each pair of disjoints. For example,  signifies that “company” is disjoint with “person”—asserting something to be a “company” commits us to asserting that it is not also a “person”. This translates to and from CLASSIC terminology as:

define-concept[person, (DISJOINT-PRIMITIVE gensym-0 person)],

define-concept[company, (DISJOINT-PRIMITIVE gensym-0 company)].

The *gensym* is necessary for translation to CLASSIC to allow linkage between the related definitions. It is lost on translation back from CLASSIC but this is not important since it is used only as a tag, not as a significant component of the knowledge base.

These basic definitions do not in themselves define the semantics of larger graphs of primitive concept nodes and subsumption or disjoint arcs. This becomes fully defined by stating that the visual language parser defines the semantics of a node in such a graph (of concept nodes only) in terms of outgoing or undirected arcs to its immediate connections only, and generates definitions as above. For convenience a single gensym is generated for each group of mutually disjoint nodes, but this does not result in different semantics from pairwise generation. Where there is a single node subsuming one group of mutually disjoint nodes, its label is used in preference to the gensym. For example, the graph:



intertranslates with the CLASSIC definitions:

```

define-concept[legal-entity, (PRIMITIVE legal-entity)]
define-concept[living_being, (PRIMITIVE living_being)]
define-concept[person, (DISJOINT-PRIMITIVE (AND legal-entity living_being) legal-
entity person)]
define-concept[government_institution, (DISJOINT-PRIMITIVE (AND legal-entity)
legal-entity government_institution)]
define-concept[company, (DISJOINT-PRIMITIVE (AND legal-entity) legal-entity
company)]
define-concept[manufacturing_company, (PRIMITIVE (AND company)
manufacturing_company)]
  
```

Note that the import-export module for the visual language makes the trivial changes necessary to the syntax of the target language, such as replacing spaces by underline. It also performs the non-trivial task of sorting the concept definitions so that terms are defined before they are used. If this cannot be done an error message is generated and the loops are visually highlighted.

2.2 Individuals

Individuals represent rigid designators in knowledge representation systems. They capture the notion of an entity having an existence in its own right such that its identity remains unchanged even if its properties are redefined. An individual is represented in the visual language as its text

label in a rectangle. For example,

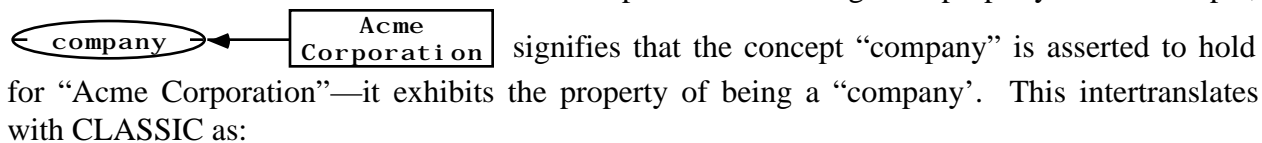
Acme Corporation

 signifies the individual “Acme Corporation”

and will continue to signify that corporation whatever properties are asserted of it and no matter how these assertions are changed. This intertranslates with CLASSIC terminology:

create-ind[Acme_Corporation].

In the visual language, the assertion that an individual exhibits a property is represented by an arrow from the individual node to a concept node encoding the property. For example,

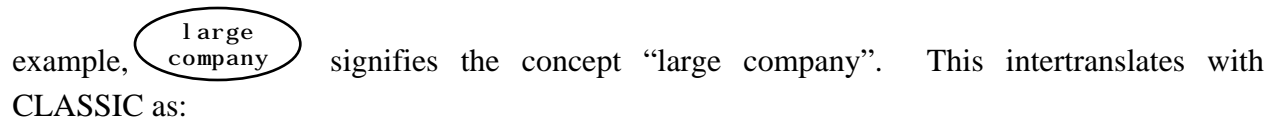


define-ind[Acme_Corporation], assert-ind[Acme_Corporation, company].

As with concepts, these basic definitions extend simply to full graphs. Examples will be given as the remaining components of the visual language are defined.

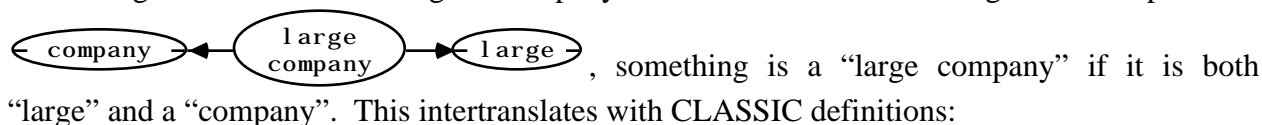
2.3 Concepts

Non-primitive concepts represent concepts that are fully defined within the knowledge representation system. They capture the notion that some concepts are fully defined by their relation to other concepts and hence do not necessarily have to be defined as subsuming other concepts or asserted as a property of an individual, but can instead be *recognized* as doing so because other definitions or assertions comply with their definition. Such a concept is represented in the visual language as its text label in an oval without marks at either end. For



define-concept[company].

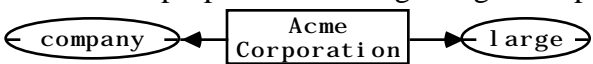
However, this definition is inadequate because no properties have been defined—anything will be recognized as a “large company”. A more meaningful example is:



define-concept[company, (PRIMITIVE company)],

define-concept[large, (PRIMITIVE large)],

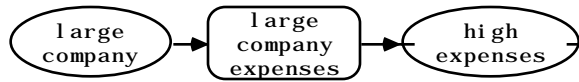
define-concept[large_company, (AND company large)].

The assertion of the properties defining “large company” as exhibited by an individual, for example , enables it to be recognized that the property of being a “large company” is also exhibited even though this has not been explicitly asserted.

2.4 Rules

Rules represent contingencies by using the recognition of an individual as exhibiting a concept to imply that another concept should also be asserted to apply to the individual. They capture those

connotations of a concept that are not deemed necessary to its definition but are deemed to apply to individuals exhibiting the properties defining that concept. A rule is represented in the visual language as its text label in a rounded corner rectangle with an arrow in from the concept to be recognized and an arrow out to the concept to be asserted. For example, the rule



signifies that when an individual is recognized as satisfying the definition of “large company” it should be asserted that it also has “high expenses”. This translates to CLASSIC terminology as:

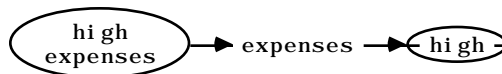
assert-rule[large_company, high_expenses].

Note that the name of the rule is lost in the translation to CLASSIC. This is because CLASSIC has a simple rule scheme that does not require rules to be identified. Sufficient intertranslatability can be achieved for CLASSIC by making the rule name a generated symbol on import, e.g. “Rule-123”. The visual language also supports a more complex rule scheme that does have a use for rule identifiers and will be discussed later.

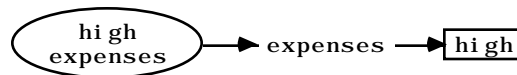
Rule translation extends simply to multiple input arrows from concepts which are treated as alternative conditions for the application of the rule, and multiple output arrows which are treated as defining a concept.

2.5 Roles

Roles provide a shorthand for expressing a group of related concepts. They capture the notion of *attributes* where the related concepts express disjoint alternatives, and *relations* where the concepts express a relation to another individual. A role is represented in the visual language as its text label without any surrounding enclosure—this corresponds to the common convention of representing roles as arc labels but gives more flexibility as roles may have multiple input and output arrows. For example, the term “high expenses” in the previous example might be better defined in terms of an “expenses” role which is constrained by the concept “high” or filled by the individual “high“:



or



The first intertranslates with the CLASSIC definition:

define-concept[high_expenses, (ALL expenses high)],

and the second with:

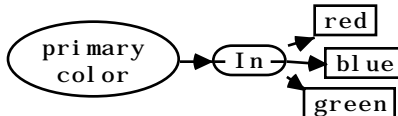
define-concept[high_expenses, (FILLS expenses high)].

Arrows coming into roles from other entities have the same semantics as arrows into concepts. Arrows out of roles define constraining concepts in the same way as arrows out of concepts define concept definitions.

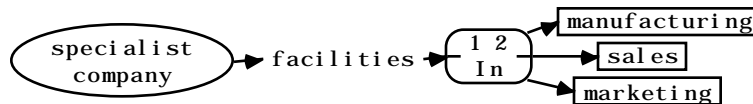
2.6 Constraints

Constraints support the residual concept-forming constraints common in term subsumption languages, such as cardinality constraints and set-inclusion constraints. They are represented in the visual language as a textual definition of the constraint surrounded by a marked rounded corner rectangle. For example, $\langle 5 \rangle$ signifies that a role has exactly 5 fillers, $\langle 1 \ 4 \rangle$ signifies that a role has a minimum of 1 and a maximum of 4 fillers, $\langle \text{In} \rangle$ signifies that a role has fillers in a set of individuals specified by outgoing arrows to those individuals, $\langle \begin{smallmatrix} 2 & 10 \\ \text{In} \end{smallmatrix} \rangle$ signifies that a role has a minimum of 2 and a maximum of 10 fillers in a set of individuals specified by outgoing arrows to those individuals, $\langle 1 \ \text{In} \rangle$ signifies that a role has exactly one filler in a set ($\langle \text{One} \rangle$ is an alternative).

Constraints are used primarily as constraints on roles, and may be used anywhere a concept would be so that they can form part of named concept definitions, for example:



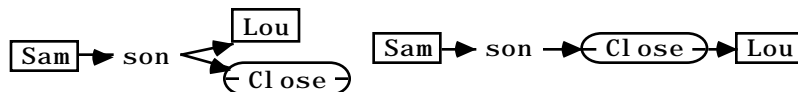
A typical use of cardinality constraints such as:



intertranslates with the CLASSIC definition:

define-concept[specialist_company, (ALL facilities (AND (AT-LEAST 1) (AT-MOST 2) (ONE-OF manufacturing sales marketing)))].

To support the open-world semantics, the constraint $\langle \text{Close} \rangle$ is used to specify explicitly that all the fillers of a role are shown. The following are equivalent:



3 Composition

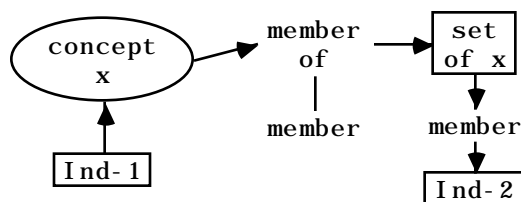
The semantics of the visual language are compositional and large knowledge structures are built up by composing smaller ones. The basic interpretation and translation mechanism is to group concept nodes with the same label together and define the concept in terms of paths along outgoing arrows from the node or nodes. This also establishes the dependencies between concepts that are used in sorting the concept definitions and detecting cycles. Individuals are analyzed in the same way except that the assertions do not need to be sorted or cycles detected. Rules are primarily analyzed as part of their incoming concepts—there is also a separate rule analysis that is not relevant to CLASSIC which is discussed later. Roles and constraints have no significance except as part of concepts and individuals, and do not need separate analysis. As a result ‘role’ text with no arrows can be used as annotation.

This ‘arrow chasing’ allows a node with the same label to occur as often or as little as is appropriate when laying out a knowledge structure to avoid crossing lines or to emphasize certain aspects of it in the visual appearance. The implementation takes advantage of this to allow knowledge structures in several different windows to be translated together so that definitions of one part of the knowledge structure can be in one window and used in another. This allows *visual libraries* to be developed supporting the re-use of knowledge from one application to another. It also allows knowledge bases to be split into components that can be separately validated and maintained.

4 Additional Features

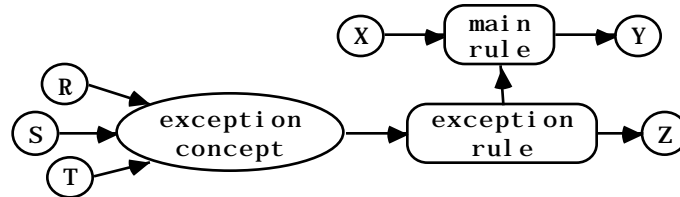
Intertranslatability with CLASSIC has been used to illustrate this paper since its KL-ONE-like syntax and semantics are well-defined and widely available. The visual language goes beyond CLASSIC in expressiveness in certain areas as already noted. The extensions in general are minor ones put in for logical completeness while preserving tractability. The complementary constraint to $\langle \text{In} \rangle$ is included since it adds to expressiveness while not causing intractability— $\langle \text{Not} \rangle$ has similar outgoing arrows to a set of individuals but expresses the constraint that fillers are not in that set. Reasoning with complementary sets is simply implemented with a flag on set constraints—union, intersection and inclusion are all well-defined. It is particularly useful in systems having open-world semantics such as CLASSIC since it allows a true open world negation to be expressed—“not red” and “one-of green or blue” express the same thing in a closed world with just those three colors but are very different concepts in an open world where more colors may be defined.

Inverse roles are also supported. In the example below “member of” and “member” are inverse relations, and the knowledge structure captures the axiom of comprehension that a concept defines a set. It is also a useful technique for collecting instances of a concept. Since the concept is non-primitive it is also defined by the set forming its extension. “Ind-1” becomes a “member” of the “set of x” by being asserted to be an instance of “concept x” and “Ind-2” is recognized as an instance of “concept x” by being asserted to be a slot filler of “member” of the “set of x”.



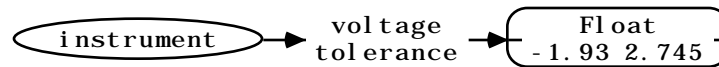
The visual language is designed to allow the representation of rules with exceptions, not only to support default reasoning, but because this form of representation usually leads to smaller, simpler and more comprehensible rule sets. It is easier to understand “when X do Y except when R S T do Z”, rather than when “X \neg R do Y and when X \neg S do Y and when X \neg T do Y and when R S T do Z”. If this representation is being used for compactness rather than to provide defaults, one does not want R, S or T being *open* to be valid reasons for either doing Z or for not preventing Y—they have to be tested and shown to fail first—one is not attempting to solve the frame problem but rather accepting the need for full exception-checking.

To represent exceptions, the visual language uses an arrow from one rule to another to specify that the second rule should be executed only if its premise succeeds *and* the first rule's premise fails (or, in default reasoning, the first rule's premise is *open* as to success/failure). This requires the recognition logic in the associated reasoning engine to be capable of distinguishing recognition success, failure (impossibility of success without retraction) and openness (possibility of success with further assertions), but this does add significantly to the computational requirements [Gaines, 1991a]. The graph below represents the example given:

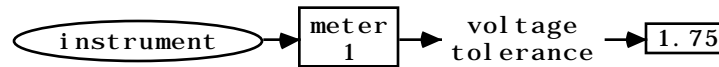


Queries are expressed in the language using $\langle - ? - \rangle$ as a query node requesting information about a concept or individual. When the results of a query are graphed inferences are separated by an infer node, $\langle \text{Infer} \rangle$. This is informative to the user and also enables the resultant knowledge structure to be edited and re-entered since the infer node adds as a block when knowledge structures are entered. An example is given in the next section.

Integer, float and date individuals are directly supported, together with corresponding conceptual constraints such as:



The inherited type constraint then makes it unambiguous that “1.75” is a numeric value rather than the name of an individual in a knowledge structure such as:



5 Knowledge Acquisition and Editing

An interactive structure editor for the visual language has been implemented as part of a knowledge acquisition toolkit. Its human-computer interaction is modeled on Apple's *MacDraw* with additional features appropriate to the language such as arcs remaining attached to nodes when they are dragged. The syntax of possible node interconnections and constraint expressions is enforced—it is not possible to enter a graph that is syntactically incorrect. Cut-and-paste of graphs and subgraphs is supported, and popup menus allow nodes to be connected with the minimum of effort. Updates are efficient and graphs with several hundred nodes can be manipulated interactively. Scroll, zoom and fit-to-size facilities allow large data structures to be navigated easily. However, partitioning data structures over several screens is encouraged and has proved practical in managing large knowledge structures.

Figure 1 shows the part of the knowledge acquisition toolkit immediately associated with the structure editor. The editor is tightly coupled to a knowledge representation server [Gaines, 1991b] that supports CLASSIC-like features extended as noted in this paper. In particular, the server computes subsumption relations between concepts and recognition of individuals by concepts. These in turn support an inference engine that fires the rules efficiently based on the

subsumption structure and the rule-to-rule links. This is coupled to a truth maintenance system which detects contradictions and supports the retraction of assertions. The server can export complete knowledge structures and the results of queries back to the graphic structure editor with automatic layout in editable form.

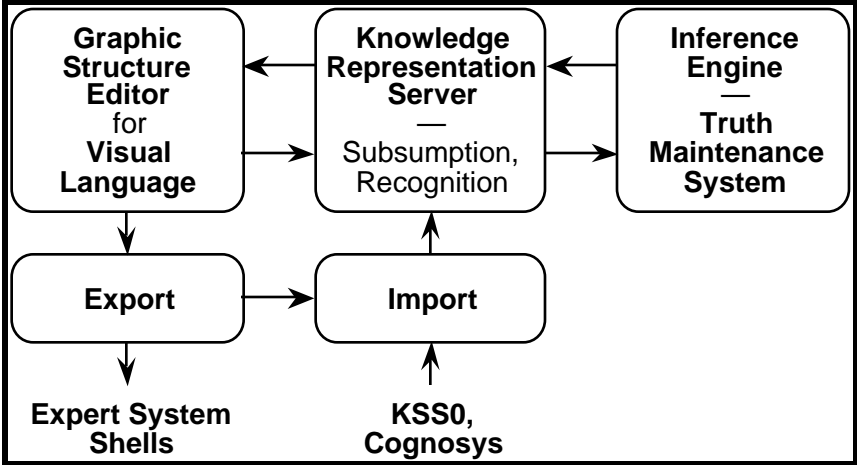


Figure 1 Graphic language editor and related sub-systems

The editor exports to a number of expert system shells and to the textual language of the knowledge representation server. The server also imports knowledge structures from other tools such as the repertory grid elicitation and induction tools in KSS0 [Gaines & Shaw, 1991] and the text analysis tools in Cognosys [Woodward, 1990], and, hence, these may be displayed for inspection, validation and editing in the graphic structure editor.

Figure 2 shows a solution to Michie’s [1989] “shuttle autolander” problem generated from exemplary cases entered through KSS0, exported to the knowledge representation server, graphed in the visual language, and reorganized and annotated for perspicuity in the structure editor.

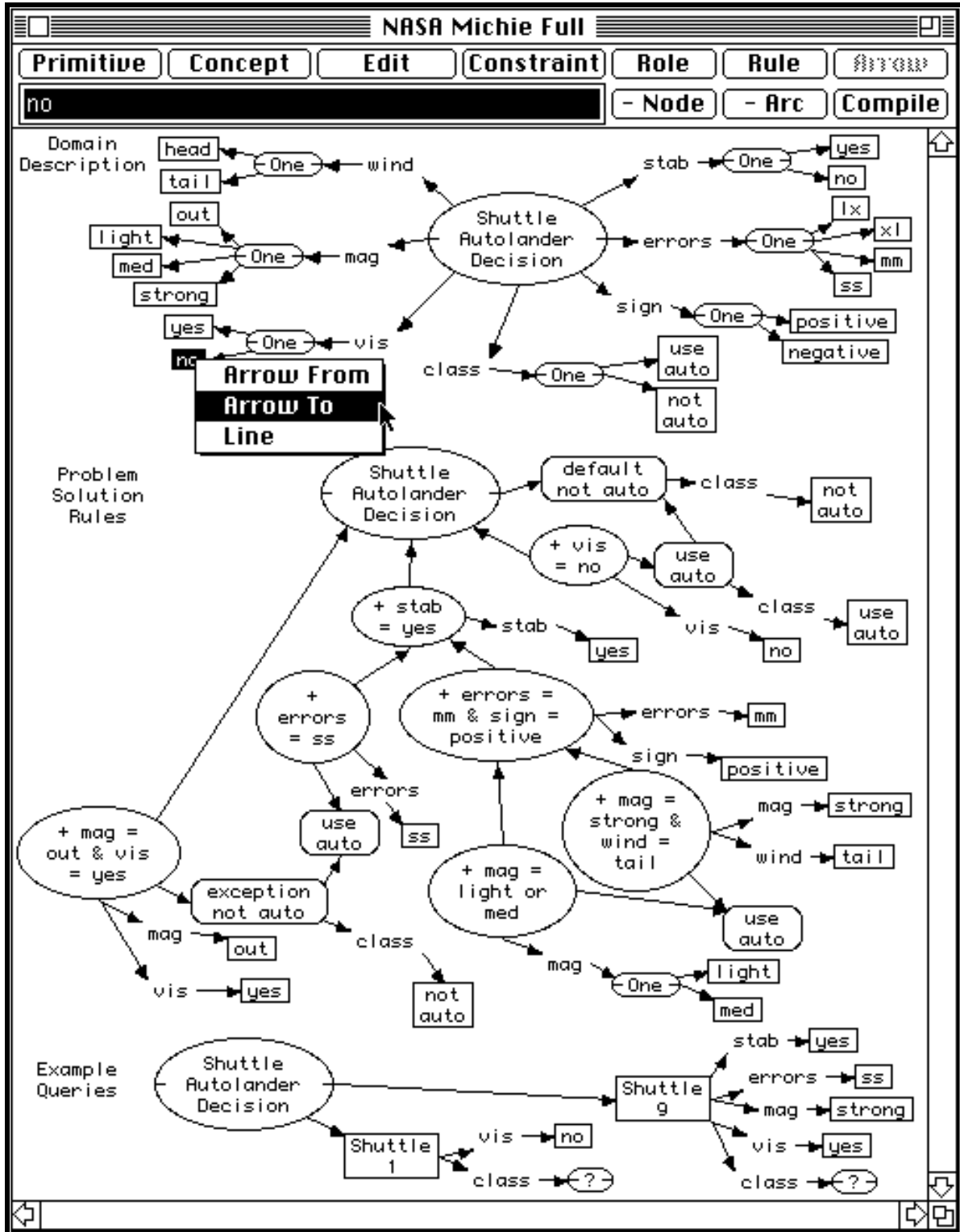
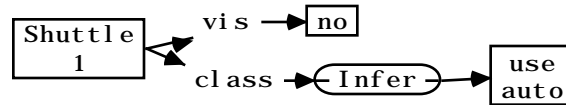


Fig. 2 Graphic language structure editor in use

At the top of the screen a descriptive concept is defined which characterizes possible shuttle situations in terms of seven attributes and their possible values. The popup menu has been activated at one of its nodes to show how arcs are entered. Below this is a set of six default rules

that solve the problem of recommending the class for a particular case. At the bottom of the screen two cases are defined with queries for the recommended values of the class role. When this query is answered graphically for the one at the bottom left, it produces the graph:



It is apparent in Figure 2 that the freedom to use the same item in several different places allows knowledge structures to be laid out informatively without visual confusion. For example, the concept “Shuttle Autolander Decision” occurs three times. It is defined at the top and the definition is used in the rules and instances. However, this separation is a matter of style—even the definition could be fragmented if the person drawing the structure felt this to be appropriate.

Similar choices have been made in the layout of the rules. For example, “exception not auto” is shown as an exception to “use auto” at the lower left, and this is itself shown as an exception to “default not auto” at the upper right. The concept forming the conclusion of “use auto” is defined at the upper right and used twice again at the lower left and right. At the lower right it is visually sensible to have two distinct concepts both connected one instance of the rule “use auto”.

Freedom in such issues of style is very significant for human understanding but has no influence on the formal interpretation of the knowledge structures.

6 Conclusions

A visual language has been presented for the representation, acquisition and editing of knowledge structures in term subsumption languages. It is a formal language in that it is syntactically and semantically well-defined and inter-translates with textual knowledge representation languages. The language is supported by an interactive graphic structure editor offering a simple and natural interface which has proved attractive to a wide variety of users.

Acknowledgements

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada. I am grateful to Ron Brachman and Rob McGregor for access to their research on term subsumption languages, and to Mildred Shaw for joint research on knowledge-based systems.

References

- [Abrett & Burstein, 1988] Abrett, G. & Burstein, M.H. The KREME knowledge editing environment. In Boose, J.H. & Gaines, B.R., Eds. *Knowledge Acquisition Tools for Expert Systems*. pp.1-24. London, Academic Press, 1988.
- [Brachman, 1977] Brachman, R.J. What’s in a concept: structural foundations for semantic nets. *International Journal of Man-Machine Studies* 9, 127-152, 1977.
- [Brachman, 1979] Brachman, R.J. On the epistemological status of semantic nets. In Findler, N.V., Ed. *Associative Networks: Representation and Use of Knowledge by Computers*. pp.3-50. New York: Academic Press, 1979.

- [Borgida *et al*, 1989] Borgida, A., Brachman, R.J., McGuinness, D.L. & Resnick, L.A. CLASSIC: a structural data model for objects. Clifford, J., Lindsay, B. & Maier, D., Eds. *Proceedings of 1989 ACM SIGMOD International Conference on the Management of Data*. pp.58-67. New York: ACM Press, 1989.
- [Brachman, Gilbert & Levesque, 1985] Brachman, R. J., Gilbert, V.P. & Levesque, H. J. An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. *Proceedings of IJCAI85*. pp.547-551. Morgan Kaufmann, 1985.
- [Brachman & Schmolze, 1985] Brachman, R.J. & Schmolze, J. An overview of the KL-ONE knowledge representation system. *Cognitive Sci.* 9(2) 171-216, 1985.
- [Cercone & Schubert, 1975] Cercone, N. & Schubert, L. Towards a state-based conceptual representation. *Proc. AAAI75*. pp.83-90. Morgan Kaufmann, 1975.
- [Fahlman, 1979] Fahlman, S.E. *NETL: A System for Representing and Using Real-World Knowledge*. Cambridge, Massachusetts: MIT Press, 1979.
- Gaines, B.R. (1991a). Integrating rules in term subsumption knowledge representation servers. *AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence*. Menlo Park, California: AAAI Press, to appear.
- [Gaines, 1991b] Gaines, B.R. Empirical investigation of knowledge representation servers: design issues and applications experience with KRS. *SIGART*, to appear.
- [Gaines & Shaw, 1991] Gaines, B.R. & Shaw, M.L.G. Eliciting knowledge and transferring it effectively to a knowledge-based systems. *IEEE Transactions on Knowledge and Data Engineering* to appear.
- [Glinert, 1990] Glinert, I.P., Ed. *Visual Programming Environments: Paradigms and Systems*. Los Alamitos, California: IEEE Computer Society Press, 1990.
- [Kindermann & Quantz, 1989] Kindermann, C. & Quantz, J. Graphics-oriented user interfaces for KL-ONE. KIT Internal Report 23. Technical University of Berlin, 1989.
- [Mackinlay & Genesereth, 1984] Mackinlay, J. & Genesereth, M.R. Expressiveness of languages. *Proc. AAAI84*. pp.226-232. Morgan Kaufmann, 1984.
- [Michie, 1989] Michie, D. Problems of computer-aided concept formation. Quinlan, J.R., Ed. *Applications of Expert Systems Volume 2*. pp.310-333. Sydney: Addison-Wesley, 1989.
- [Nosek & Roth, 1990] Nosek, J.T. & Roth, I. A comparison of formal knowledge representations as communication tools: predicate logic vs semantic network. *International Journal of Man-Machine Studies* 33, 227-239, 1990.
- [Schmolze, 1983] Schmolze, J. KLONEDRAW—a facility for automatically drawing pictures of KL-ONE networks. *Research in Knowledge Representation for Natural Language Understanding*. pp.41-44. Report No.5421, Cambridge, Mass.: Bolt Beranek and Newman, 1983.
- [Watanabe, 1989] Watanabe, H. Heuristic graph displayer for G-BASE. *International Journal of Man-Machine Studies*, 30(3) 287-302, 1989.
- [Woods, 1975] Woods, W.A. What's in a link: Foundations for semantic networks. Bobrow, D.G. & Collins, A.M. (Eds) *Representation and Understanding: Studies in Cognitive Science*. pp.35-82. New York: Academic Press, 1975.

[Woodward, 1990] Woodward, B. (1990) Knowledge engineering at the front-end: defining the domain. *Knowledge Acquisition*, 2(1) 73-94, 1990.