IJCAI-79

第 6 回人工知能国際会議

Proceedings of the Sixth International
Joint Conference on Artificial Intelligence
Tokyo, August 20–23, 1979
Volume Two

AN EXAMINATION OF A FRAME-STRUCTURED REPRESENTATION SYSTEM

Mark Stefik
Computer Science Department
Stanford University
Stanford, California 94305

The Unit Package is an interactive knowledge representation system with representations for individuals, classes, indefinite individuals, and abstractions. Links between the nodes are structured with explicit definitional roles, types of inheritance, defaults, and various data formats. This paper presents the general ideas of the Unit Package and compares it with other current knowledge representation languages. The Unit Package was created for a hierarchical planning application, and is now in use by several AI projects.

1 INTRODUCTION

When the MOLGEN* project was started, many ideas about frame systems and semantic networks were being widely discussed, but no software was available. In collaboration** with other members of the MOLGEN project, the author developed a representation system called the "Unit Package" which became operational in July 1977. In some cases (and usually in ignorance), this work has duplicated other representation work that was happening at about the same time. The Unit Package is now being used by several other projects including two away from Stanford. It is written in INTERLISP and runs under the TENEX and TOPS20 operating systems. It is an interactive system for building knowledge-based programs. It also provides a substantial virtual memory so that knowledge bases of several thousand nodes can created without sacrificing the INTERLISP environment.

*MOLGEN is a joint project between the Stanford Computer Science Department, several departments of the Stanford Medical School, and the Computer Science Department at the University of New Mexico. It is active in the computer planning of molecular genetics experiments ([7], [17]). MOLGEN is supported by grant NSF MCS 78-027777 from the National Science Foundation.

**Nancy Martin and Peter Friedland made substantial contributions to the Unit Package, especially in the first year.

2 ELEMENTS OF THE REPRESENTATION

Knowledge in the Unit Package is organized as a partitioned semantic network. Following KRL [3] terminology, the nodes are alternatively called units and the links are called slots. A built-in generalization relationship provides a hierarchy with several modes of property inheritance. Conspicuous for its absence is a comprehensive inference mechanism. While the Unit Package provides some built-in inferential facilities — notably the property inheritance mechanism, the pattern matchers, and the attached procedure mechanism — most of the inference control must be provided by application-specific methods. The ideas in the Unit Package will be presented in the following order.

1) Partitions — The boundaries in the network which divide it into (possibly overlapping) explicit sets of nodes.

2) Units — The nodes in the network. The Unit Package has nodes for constant individuals and classes as well as for undetermined and abstract entities. For the latter it permits naming them, anchoring them to constants, adding details to them, and indicating whether they are the same or different (without necessarily anchoring them).

3) Slots — The links between the nodes. The fine structure of links is defined by a set of "aspects" which define their inheritance and definitional roles, datatypes, defaults, and attached procedures. These aspects provide declarative meta-knowledge which indicates how a slot is to be interpreted.

4) Attached procedures — Procedures may be attached to units, slots, or datatype units, depending on what they are used for. They are activated by messages and have explicit purposes. A small set of purposes is recognized by the Unit Package to allow automatic activation under specified circumstances. Attached procedures are the chief mechanism for making inferences in the Unit Package.

2.1 Partitions

While a knowledge base is ultimately composed of nodes and links, it is often useful to consider larger organizations of units. For this purpose the Unit Package provides a facility for partitioning a network into explicit sets. This is the same idea as the spaces in Hendrix's* partitioned semantic networks [9]. The Unit Package simply provides an efficient notation for explicit sets** and a number of functions which act on all members of sets.

2.2 Nodes

In our planning application, it is helpful to have open-ended descriptions for abstract entities and to be able to refer to individuals whose identities have not yet been determined. To represent this information the Unit Package provides four kinds of nodes as in Fig. 1 along with the computational machinery for their interpretation.

*We have not yet exploited the set notation for handling quantification.

**This could have been indicated with a member-of slot. Our implementation makes it possible to tell whether a unit is in a set without requiring that the unit be core resident.

	Constant	Variable
Individual	Instance	Indefinite
Class	Schema	Description

Figure 1. Kinds of nodes in the Unit Package

These kinds of nodes are listed here briefly and discussed in the following sections.

1) INSTANCE — Constant node that stands for a unique individual. An instance is analogous to a constant in predicate calculus.

2) SCHEMA — Constant node that stands for a class. A schema describes all of its potential progeny in the generalization hierarchy.

3) INDEFINITE NODE — Variable node that stands for an individual. Analogous to an existentially quantified variable in "predicate calculus with equality". Fundamental links for reasoning about equality are (1) "anchor" links which tie indefinite nodes to instances and (2) "co-reference" and "not-co-reference" links which tie indefinite nodes together.

4) DESCRIPTION NODE — Variable node that stands for a class. These nodes are used for matching and reasoning about abstractions, that is, incompletely described entities.

Section 3.1 compares these node types to other work on representation.

2.2.1 Nodes for Constants

An "instance" is the simplest kind of node in the Unit Package. Instances are analogous to constant terms in predicate calculus and are used to represent distinct entities, that is, "individuals" Examples of instances in the our application would be domain objects like Culture-133, a particular culture of organisms. Schemata stand for classes, that is, implicit sets and may have subclasses and instances below them in the hierarchy. Because instances do not stand for classes, they can have no nodes below them. A schema describes all of its potential progeny by expressing all of the attributes that are required for members of the class.

2.2.2 Indefinite Nodes

"Indefinite" nodes are the simplest variable nodes in the Unit Package; they stand for individuals whose identity may be unknown. They allow us to indicate that two variables are equal (or not equal) without knowing their values. This is like the augmentation of predicate calculus to predicate calculus with equality except that it is a three-valued logic permitting the logical possibility that equality may be "unknown". For example, we could have indefinite nodes for Martha's-husband or The-first-president. We may equate these indefinite nodes to each other by marking them as "co-referential". We may also "anchor" or ground them to an individual unit (e.g., George-Washington). Two indefinite nodes are said to be equal if they have the same anchor or if they are tagged as being co-referential. Two indefinite nodes are not equal if they have different anchors, if they are marked as "not co-referential", or if their scopes do not intersect*. In all other cases, equality is "unknown". Co-reference and anchor relationships are represented by slots with standard names. The need for such nodes in natural language processing (sometimes termed "intentional" nodes) has also been recognized [20]. Our experience has been that the potentially difficult parts in reasoning with these entities are (1) deciding when there is enough evidence to conclude about equality of identities and (2) combining the (possibly conflicting) attributes of the various hypothetical objects once their common identity has been concluded.

2.2.3 Description Nodes

Description nodes are variable nodes which stand for classes instead of individuals. These nodes are used in our planning application to represent open-ended goals. Whereas "anchoring" is the key to equality of indefinite nodes, matching is the key to equality of description nodes. The class defined by a schema is the set of its specializations in the hierarchy; the class defined by a description node is the set of nodes within a specified scope that match the description node. Two description nodes are said to be equal if they potentially match the same units* or if they are indicated as being co-referential. This limits the expression of match conditions to a conjunction of slots. We have found it convenient in our planning application to augment this notation with additional "constraint" units that can express arbitrary predicates involving several nodes. In our planning application, the refinement of abstract goals is accomplished by the addition of constraints to descriptions and the matching of descriptions to the knowledge base. The interpretation and manipulation of these constraints is done by our planning program and not by the Unit Package.

*Scopes, in the Unit Package, are simply a branch of the generalization hierarchy. Extending this to a more flexible form of quantification is part of the planned future work on the Unit Package.

2.3 Links

As the Unit Package has been developed, it has become necessary to provide structure for the links between the nodes. Several different "aspects" have been created to accommodate this:

1) NAME — Name of the slot.

2) VALUE — Stored value of the slot. Typically this is the name of a unit but it may be a different data-structure if the datatype aspect is other than "unit". The Unit Package distinguishes between values that are value-restrictions and those that are "terminal"**

3) DEFINITIONAL ROLE — The role that the slot plays in the definition of the unit. The roles "PART-OF", "PROPERTY", "RELATION", "SUPER-UNIT", "EQUIVALENCE", and "DOCUMENTATION" are recognized by the Unit Package. Definitional roles are explained further in Section 2.3.1.

4) INHERITANCE ROLE — The mode of inheritance of the slot by progeny. Four distinct roles, ("S", "R", "O", and "U") are recognized in the Unit Package. Criteriality is determined from this role. Inheritance roles are discussed further in Section 2.3.2.

*The operational test for deciding whether two descriptions potentially match the same units is that the descriptions match each other.

**In an analogy with units, descriptions are like schemata and terminal values are like instances.

5) DEFAULT — Default value for the slot to be used in the absence of specific information. This aspect was much less frequently used than we originally expected.

6) DATATYPE — A link in the Unit Package need not go to another unit; it may go to an atom, integer, string, list, lambda expression, or to a value with some other "datatype". "Unit" is probably the most important datatype and is used for most of the links usually associated with semantic networks. The datatype aspect facilitates treating different kinds of values uniformly. The factoring of procedures to support this is discussed in Section 2.4.

2.3.1 Definitional Roles

A retrospective examination of how slots were being used in our genetics knowledge base made us aware that indistinguishable notations were sometimes used where distinct meanings were intended. Fig. 2 illustrates some of the slots from the representation of an organism in one of our knowledge bases. The slot chromosome is used to refer to a part of the organism; the slot gramstain refers to the visible character of the organism when a particular staining agent is applied. Without the definitional role, a general procedure to separate or remove the parts of a unit may try to "remove the gramstain" from an organism. Thus, the exosomes slot in Fig. 2 indicates (1) what exosomes the organism has (i.e. EXOSOMES (ORGANISM1) = (PMB9, PSC1Ø1)) and (2) that the exosomes are to be considered parts of the organism (i.e. PART-OF (PMB9, ORGANISM1) = T).

The idea of distinguishing between kinds of links is not new with our work. Woods [2Ø] distinguished between "assertional" and "structural" links. We differentiate between the definitional roles "PART-OF", "PROPERTY", "RELATION", "SUPER-UNIT", "EQUIVALENCE", and "DOCUMENTATION". The role "SUPER-UNIT" has the inverse meaning of "PART-OF" and is used to annotate slots which point back from the parts of a larger concept. The role "EQUIVALENCE" is used to label the special "co-reference" and "anchor" slots used in description and indefinite units. The role "DOCUMENTATION" is used to tag slots like the modifier slot which are part of the automatic documentation of a knowledge base. More definitional roles will probably emerge as the Unit Package is used in additional applications.

Slot Name	Value	Definitional Role
EXOSOMES:	(PMB9, PSC1Ø1)	PART-OF
CHROMOSOME:	DNA-Structure-92	PART-OF
MORPHOLOGY:	COCCUS	PROPERTY
GRAMSTAIN:	POSITIVE	PROPERTY
CAN-EAT:	NUTRIENT-GEL	RELATION
KILLED-BY:	TETRAMYCIN	RELATION
MODIFIER:	Feitelson	DOCUMENTATION
MODIFIED:	12-MAR-79	DOCUMENTATION

Figure 2. Definitional Roles

2.3.2 Inheritance Roles

The idea of the hierarchical inheritance of properties has its roots in the inferential machinery discussed in Quillian's thesis [14]. In its simplest form, a value is defined in the most general schema to which it applies. Nodes corresponding to descendants of the schema inherit the value. As Brachman [6] points out, while a slot in an individual is intended to assert a property of the individual, a slot in a schema is often intended to restrict the legal values of corresponding slots in its potential progeny. These alternative meanings of slots (slots used to define properties versus slots used to instantiate properties) are important for the understanding of inheritance and are not always clearly differentiated in network formalisms.

In the Unit Package we have identified five standard "inheritance roles" for slots. Four roles (termed "S", "R", "O", and "U") that have been useful in the Units Package are described below. A fifth role (termed "M"), which is probably useful but which has not been implemented, is also discussed.

1) The simplest role is for direct inheritance of values by all progeny. This is termed the "S" role because the slot has the same value in the defining unit and all of its progeny.

2) The "R" role is for the inheritance of requirements; it is for slots that are criterial to the definition of a schema. The

values in schemata are interpreted as value-restrictions for the corresponding slots in progeny. The values in slots of progeny need not be the same as in the schema, but may be further restricted*. Usually the value of the slot in an instance will be "terminal", that is, an actual value instead of a description of a value. If every "R" slot in an instance has a terminal value, then the instance is fully instantiated.

3) The "O" role is similar to the "R" role except that the slot is optional (ie. not criterial). The slot in an instance need not be filled in with a terminal value. For example, in the molecular genetics domain, the organisms slot of the Culture unit has role "R" and the exosomes slot of the Bacterium unit has role "O". This is meant to indicate that every culture must have organisms, but every bacterium need not have exosomes.

4) The "U" role is for information about a node that is not inherited by its progeny. The slot value is unique at each level in the hierarchy. As with the other roles, the slot name, datatype, and role are inherited by offspring. However, the value is not inherited or used to limit the value of the slot in offspring. This role is used mainly for our bookkeeping purposes in the network such as for the creator slot in units.

The first three roles can be seen as decreasingly strict. With the "O" role, a value is optional in instances; with "R" it is required; with "S" it is directly inherited. In contrast, the "U" role ignores the nesting of values in progeny altogether. One additional role for inheritance, which we have not implemented, would allow slots in progeny to have values that override restrictions from above. (This might be termed the "M" role meaning modifications allowed). This idea was originally disallowed because it was seen as counter to the spirit of unit specialization.

Like definitional roles, inheritance roles provide meta-knowledge about the interpretation of slots. They indicate (1) transmission instructions for the information and (2) whether the value is criterial to the definition of the slot. Inheritance roles are also used by the interactive component of the Unit Package to determine what checking is performed when knowledge is entered by a user.

Before leaving the subject of inheritance, it is worth emphasizing that a generalization hierarchy should be distinguished from other hierarchical relationships from everyday life which do not indicate inheritance paths. Some common ones are "subset-of", "element-of", "part-of" and "abstraction-of". These relationships are represented in a semantic network by explicit links independent of those used for the superclass taxonomy.

2.4 Attaching Procedures

The attachment of procedures to frames is one of the representational ideas common to the new knowledge representation languages. Several older languages (such as LISP) support arbitrary attachment of procedures to data structures; the approach in frame-structured languages is more disciplined. This discipline in the Unit Package has two main elements: (1) standardized points for attachment and (2) an explicit purpose for procedures. The "purpose" of a procedure indicates when a procedure should be activated.

The Unit Package provides three standard places for attaching procedures: (1) unit attachment (2) slot attachment and (3) datatype attachment. Unit attachment is used for operations that act on a unit as a whole. Slot attachment is used for operations on a particular slot of a unit. Datatype attachment is useful for operations on slots located in units throughout the knowledge base that have values of a particular datatype. Although datatype attachment has not been reported in other frame systems, it has been used extensively in the Unit Package [17] and is similar to ideas from SIMULA.

We have adapted some terminology from SMALLTALK to describe the activation of attached procedures. A procedure is activated by "sending it a message" with a token that matches its purpose. This is an indirect form of procedure call — where the purpose of the procedure is known to the caller but the name of the procedure need not be known. The "purpose" of a procedure is usually associated with standardized activation conditions (e.g. "To-Get"); it also indicates where the

*The meaning of "further restricted" is determined by a function associated with the datatype of the slot. For the "number" datatype, the value-restrictions are numeric ranges or lists. For the "unit" datatype, the value-restrictions may be description nodes or ancestor specifications.

procedure's name is stored. For unit attachment, the purpose is the name of a slot containing the procedure name; for slot attachment, it is the name of an aspect; for datatype attachment, it is the name of the slot in the unit for the datatype. Functions like the matcher, which must work for all datatypes, operate by activating datatype-specific procedures using the message mechanism.

3 RELATIONSHIP TO OTHER WORK

During the period when the Unit Package was developed, many other frame-structured representation systems have also appeared. In some cases, our work has duplicated that of other groups. This section compares the Unit Package to related systems.

3.1 Other Work on Node Types

The idea of distinguishing between node types in a knowledge representation language has important implications for the design of interpreters and matchers. If node types are not distinguished, then the semantic information about abstractness and equality must be carried by some other means. Node types similar to our four categories have appeared in other contemporary representation languages. Fig. 3 summarizes what they are called in AIMDS ([15], [16]), FRL [8], KLONE ([5], [19]), KRL-Ø ([1], [3]), Levesque's System [11], and OWL ([18],[13]). This is not to say that the node types correspond exactly in these systems, but that the ideas seem to be very similar.

Several other types of nodes have appeared in representation languages. For example, KRL-Ø supported the additional node categories "relation", "proposition", and "basic". The relation category was used to represent an abstract relationship between entities and a proposition was an instantiation of a relationship specifying its truth value. (In our planning application, constraints serve some of this function although their interpretation is by the planning program instead of a general interpreter for the representation language.) "Basic" categories partition the world into simple non-overlapping categories. Categories are meant to allow quick tests (by category comparison) to decide whether an object fits a description. This partitioning is partially (but inadequately) achieved in the simple hierarchical systems by the approximate rule that two schemata are in distinct categories if neither is an ancestor of the other in the superclass hierarchy*. It is not clear that such categories can be assigned to domain objects once and for all. Further aspects of this are discussed in two recent critical examinations of KRL ([10], [1]).

*One of the shortcomings of the Unit Package is illustrated by this example. There is no annotation to indicate which branches in the generalization hierarchy are mutually exclusive.

System	Instance	Schema	Indefinite	Description
AIMDS	instance	template		
FRL	instantiated frame	generic frame		
KLONE	individual nexus	generic concept	variable nexus	parametric individual
KRL-Ø	individual	specialization	manifestation	abstract
Levesque's System	instance	class	indefinite	
OWL	instance	species		

Figure 3. Node types in other representation systems

The node types in KLONE ([4], [5]) shown in Fig. 3 do not fit perfectly into the categories used in the Unit Package; KLONE distinguishes between patterns (also called descriptions or concepts) and representations of things. A generic concept in KLONE is a pattern because it does not stand directly for its potential progeny; the class is its extension in some context. Similarly, an "individual concept" in KLONE is an individualized pattern; it describes a potentially unique individual which may or may not exist (e.g. "the current king of France"). A nexus is more like the sort of individual in the Unit Package. A nexus is either constant or variable. A nexus has an "existence value" which can be "does not exist". It can have "coreference wires" to other nexuses and "description wires" to individual concepts. Thus KLONE distinguishes individualized patterns (individual concepts) from representations of individuals (nexuses). The matrix in Fig. 1 does not make this distinction. KLONE's "parametric individual" is not quite the same as a description node (i.e. variable class node) in the Unit Package. Parametric individuals are still patterns — they describe potentially many individuals — but they are not arbitrary generic descriptions; some of their values are bound by the context in which they appear, and thus they are "parameterized".

3.2 Other Work on Property Inheritance

The recognition of different forms of inheritance is not unique to the Unit Package. Goldstein and Roberts [8] distinguished two kinds of inheritance in FRL-Ø — additive and restrictive. Additive inheritance permitted a specialization to add new non-contradictory facts. This corresponds to the "R" and "O" roles above. Restrictive inheritance is used when a specialization overrides the information in a more general schema. This corresponds to the "M" role above, which we have not implemented. FRL-Ø also encouraged the use of "idiosyncratic forms" of inheritance by the use of attached procedures. These procedures effected the transmission of values from frames other than the parent. This seems to be an attempt to capture some of the "multiple-perspectives" idea of KRL using less machinery.

An elaborate proposal inheritance was proposed by Brachman [6] and then substantially elaborated by Brachman and Woods ([5], [19]). Brachman connects nodes not by a single generalization link, but rather by a complete bundle of links (called a "cable") which ties together the parts with different kinds of links. The details of these links require more explanation than is convenient here but the main insight is that inheritance of complex structures must be supported by a rich vocabulary of relationships for the parts of the inherited structure. While the scope of their developing system (KLONE) is broader than our own, the philosophy is essentially the same, namely, to develop a concise and structured representation in which the important kinds of relationships are explicit and uniformly represented for processing by general purpose network routines.

4 SUMMARY

The Unit Package is a frame-structured representation language whose main elements are partitions, nodes, links, and attached procedures. Our goal has been to develop a concise and structured representation language in which the important kinds of relationships are explicit and uniformly represented for processing by uniform network processing routines. This has led to formalisms for kinds of nodes, links, inheritance, and attached procedures. Some of these formalisms have been inspired by and instrumental for the MOLGEN planning application. Probably the most important example of this is the idea of a hierarchical variable to which one can add information and constraints. These variables provide a notation for hierarchical planning.

Some shortcomings of the Unit Package have been noted in passing. These include a need for allowing multiple generalizations, a need for explicit indication of mutual exclusion, and a more adequate use of quantification. These problems are not difficult to fix. Space does not permit discussion of more substantial representational problems beyond the current research, such as the structured representation of processes and causal relationships. The reader interested in a discussion of some unsolved problems is referred to [17].

5 ACKNOWLEDGMENTS

Many thanks to Avron Barr, Jim Bennett, Bruce Buchanan, Ed Feigenbaum, Peter Friedland, Jonathan King, and Doug Lenat who read earlier drafts of this document and whose criticisms have greatly improved its readability. Thanks also to the other members of the MOLGEN project who have helped in this work and who have shown great patience while other needs in our project have been pressing. Computing support was provided on the SUMEX-AIM facility by a grant from the Biotechnology Resources Branch of the National Institute of Health.

REFERENCES

1. Bobrow D.G., Winograd T., KRL, Another Perspective, Cognitive Science 3 29-42 (1979)

2. Bobrow D.G., Winograd T., Experience with KRL-0, One cycle of a knowledge representation language, Proc. IJCAI-77, MIT, Cambridge, Mass. August, 1977, pp. 213-222.

3. Bobrow D.G., Winograd T., An Overview of KRL, a Knowledge Representation Language, Cognitive Science 1:1. (1977) 3-46.

4. Brachman R., Personal communication, (May 1979)

5. Brachman R., Theoretical studies in natural language understanding, BBN report 3888 (September 1978)

6. Brachman R., What's in a concept: Structural foundations for Semantic Networks, BBN report 3433, (October 1976)

7. Friedland P., Knowledge-based Experiment Design in Molecular Genetics, in Proc. IJCAI-79 (this volume)

8. Goldstein I.P., Roberts R.B., NUDGE, A knowledge-based scheduling program, in Proc. IJCAI-77, MIT Cambridge, Mass., August, 1977, pp. 257-263.

9. Hendrix G.G., Expanding the Utility of Semantic Networks through Partitioning, SRI Technical Note 105 (June 1975)

10. Lehnert W., Wilks Y., A critical perspective on KRL, Cognitive Science 3:1 1-28 (1979)

11. Levesque, Mylopoulos, McCalla, Melli, and Tsotsos, A formalism for modelling, Proc. First CSCSI/SCEIO National Conference, Vancouver, 1976, pp 243-254.

12. Martin N., Friedland P., King J., Stefik M., Knowledge Base Management for Experiment Planning, in Proc. IJCAI-77, MIT, Cambridge, Mass., August, 1977, pp. 882-887

13. Martin W.A., OWL, in Proc. IJCAI-77, MIT, Cambridge, Mass., August, 1977 pp. 985.

14. Quillian R., Semantic memory, in Minsky M. (ed.) Semantic Information Processing Cambridge: MIT Press (1968)

15. Sridharan N.S., AIMDS User Manual - Version 2., Rutgers University Computer Science Technical Report CBM-TR-89. (June 1978)

16. Sridharan N.S., Knowledge representation in AIMDS and its use in BELIEVER, in Proc. IJCAI-77, Cambridge, Mass., August, 1977, pp. 990

17. Stefik M.J., Orthogonal Planning with Constraints, Report on a knowledge-based program that plans synthesis experiments in molecular genetics, (forthcoming dissertation) Computer Science Department, Stanford University (September 1979)

18. Szolovits P., Hawkinson L.B., Martin W.A., An overview of OWL, a language for knowledge representation, Technical Report MIT/LCS/TM-86 from Laboratory for Computer Science at MIT (June 1977)

19. Woods W.A., Brachman R.J., Research in Natural Language Understanding, Bolk Baranek and Newman Technical Report 3963 (August 1978)

20. Woods W.A., What's in a link: Foundations for Semantic Networks, in Bobrow D.G., Collins A. (eds.), Representation and Understanding, Academic Press (1975)