AD-A041 372 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 5/7
AN OVERVIEW OF OWL, A LANGUAGE FOR KNOWLEDGE REPRESENTATION.(U)
JUN 77 P SZOLOVITS, L B HAWKINSON, W A MARTIN N00014-75-C-0661
UNCLASSIFIED MIT/LCS/TM-86 NL

1 OF 1

AD
A041372

END

DATE
FILMED
7-77

12

ADA041372

LABORATORY FOR
COMPUTER SCIENCE
(formerly Project MAC)

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-86

AN OVERVIEW OF OWL,
A LANGUAGE FOR KNOWLEDGE REPRESENTATION

Peter Szolovits
Lowell B. Hawkinson
William A. Martin

DDC
JUL 8 1977
D

June 1977

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

This work was supported by the Advanced Research
Projects Agency of the Department of Defense and
was monitored by the Office of Naval Research
under Contract No. N00014-75-C-0661

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

AD No.
DDC FILE COPY

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
(14) MIT/LCS/TM-86

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (*and Subtitle*)
(6) An Overview of OWL, A Language for Knowledge Representation,

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER
MIT/LCS/TM-86

7. AUTHOR(s)
(10) Peter Szolovits, Lowell B. Hawkinson and William A. Martin

8. CONTRACT OR GRANT NUMBER(s)
(15) N00014-75-C-0661

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Massachusetts Institute of Technology
Laboratory for Computer Science
545 Technology Square
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
(12) 29p.

11. CONTROLLING OFFICE NAME AND ADDRESS
Advanced Research Projects Agency
Department of Defense
1400 Wilson Boulevard
Arlington, Virginia 22209

12. REPORT DATE
(11) June 1977

13. NUMBER OF PAGES
29

14. MONITORING AGENCY NAME & ADDRESS (*if different from Controlling Office*)
Office of Naval Research
Department of the Navy
Information Systems Program
Arlington, Virginia 22217

15. SECURITY CLASS. (*of this report*)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

5/8 5/ [illegible]

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Artificial Intelligence	OWL
Knowledge Representation	Symbol Manipulation
LMS	
Memory Structures	
Natural Language	

is described

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

We describe the motivation and overall organization of the OWL language for knowledge representation. OWL consists of a memory of concepts in terms of which all English phrases and all knowledge of an application domain are represented, a theory of English grammar which tells how to map English phrases into concepts, a parser to perform that mapping for individual sentences, and an interpreter to carry out procedures which are written in the same representational formalism. The system has been applied to the study of interactive

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

409 648

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

20. dialogs, explanations of its own reasoning, and question answering.

ACCESSION for
NTIS White Section ☑
DDC Buff Section ☐
UNANNOUNCED ☐
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

12

MIT/LCS/TM-86

An Overview of OWL,
A Language for Knowledge Representation

Peter Szolovits
Lowell B. Hawkinson
William A. Martin

June, 1977

D D C
JUL 8 1977
D

Massachusetts Institute of Technology

Laboratory for Computer Science

(formerly Project MAC)

Cambridge Massachusetts, 02139

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Abstract

We describe the motivation and overall organization of the OWL language for knowledge representation. OWL consists of a memory of concepts in terms of which all English phrases and all knowledge of an application domain are represented, a theory of English grammar which tells how to map English phrases into concepts, a parser to perform that mapping for individual sentences, and an interpreter to carry out procedures which are written in the same representational formalism. The system has been applied to the study of interactive dialogs, explanations of its own reasoning, and question answering.

Keywords:
Artificial Intelligence, Knowledge Representation, LMS, Memory Structures, Natural Language, OWL, Symbol Manipulation

Acknowledgements

This work was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract •N00014-75-C-0661.

We wish to thank our colleagues and students in the Laboratory for Computer Science and the Artificial Intelligence Laboratory at MIT for their contributions to our understanding.

This paper was presented at the Workshop on Natural Language for Interaction with Data Bases held by the International Institute for Applied Systems Analysis (IIASA) at Schloss Laxenburg, Austria, in January, 1977. It will also appear in the proceedings of that Workshop [Rahmstorf 1977].

Table of Contents

I. Overview and Motivation 1

II. The Linguistic Memory System 5

A. The Essential Structure of Concepts 5
B. Attachment 7

III. How English Phrases are Represented as Concepts 10

A. What Does an English Expression Say? 11
B. Kinds of Specialization 14
C. Parsing 17

IV. Reasoning 20

Bibliography 23

I. Overview and Motivation

We have undertaken the design and implementation of a new computer language for *knowledge representation*, called OWL. We have become convinced that recent progress in Linguistics and in *Artificial Intelligence* (AI) now suggests a set of principles which are worth implementing as part of a programming language to make them uniformly accessible for our further work.

For a computer program--as for a person--*it is more effective to know how to do something than to be able to figure it out.* The AI field has made important progress under an opposite set of assumptions: that all knowledge of the domain should be expressed in propositional form and that a program's actions should be directed by a general-purpose problem solver operating on propositions representing the application world. Such a problem solver would always figure out what to do next based on the state of the world and its set of general principles. At the same time, most programs which have been used for their ability to perform in an application domain rather than for their pedagogic clarity have used a very different form of organization: the knowledge of how to perform the task was implicitly built into the steps of the program. Of course, such an organization is generally accompanied by inflexibility, difficulty of extension, incomprehensibility and unprovability of the program, and many other ills. If, however, we could express the description of the procedural knowledge of the program in the same formalism as its declarative knowledge of the domain of application, then both would be equally accessible.

This is precisely what is done in OWL--the program is just another aspect of the description of the application world, and knowledge of how to solve specific problems of that world can be explicitly embedded in the description.

We have taken English as the basis for our knowledge representation formalism. The greatest attraction of this approach is that it almost trivially satisfies our need for expressive power. After all, native speakers of English can usually communicate their knowledge of any domain of interest in English,[1] perhaps augmented by specialized notations and vocabularies particular to the domain. Because we choose a computer representation which is designed to be similar to the natural language employed by a computer-naive user of one of our programs, we expect that the translation process from English sentences to our internal structures will be straightforward. Once we succeed in translating the English phrase into our internal representation, that will allow *all* of OWL's activities, including understanding the sentence in semantic detail, resolving references, mapping the sentence onto some capability of the system for acquiring new knowledge or answering on the basis of old, etc., to make use of the same representational formalism. This, in turn, will help us to make the complete operation of the program accessible for explanation to, and modification by, someone who may well understand the domain of application but not our computer technology.

---

[1] We limit ourselves to "left-hemisphere knowledge," which does not include visual skills or manipulative skills where local muscle/nerve training is an essential component. Thus, our domains are restricted to reasoning tasks where the necessary data about a problem can be acquired verbally; e.g., medical diagnosis and treatment of the type which could be done by consultation over the telephone (probably not, for example, diagnosis of skin disease, where visual inspection is a critical skill), automatic program writing, question answering.

Arguments for English as a programming language have been made since the early 1960's, yet it has not been universally acclaimed as desirable. The principal objections to basing a programming language on English (or any natural language) center on the innate ambiguity of natural language and its lack of conciseness when contrasted with special mathematical notations. The second problem is rapidly resolved if we extend our definition of *natural language* to allow the incorporation of new notations. After all, the natural language of a physics text is hardly the literary English of the day. The first problem has both a trivial and a difficult component: pure syntactic ambiguity, as created by the existence of homonyms for instance, is simply controllable, whereas ambiguity arising from the fact that what one (literally) says is not what one actually means is, of course, difficult. Our response is simply that we wish to begin by representing precisely what one says, and we will allow the determination of the meaning of each utterance to be part of the problem that the system is to solve.

During the past few years, we have implemented the following components of a complete system based generally on the above ideas:

A *Linguistic Memory System* (LMS) [Hawkinson 1975], which is a memory (data base) of *concepts* in which all knowledge in OWL resides. LMS can be viewed as a semantic network, with a somewhat unusual interpretation of its nodes and arcs.

A theory of English grammar which specifies how any utterance of English can be represented in terms of LMS concepts.

A skeletal world model, organized as a taxonomy of concepts, and intimately related to the theory of English grammar.

An augmented transition network parser to translate English utterances into their OWL representations.

A generator to perform the inverse transformation to the parser.

An interpreter which carries out procedures represented in the OWL formalism.

An explainer which provides English explanations (via the generator) of procedures and data dependencies known to the interpreter, as well as results of previous executions of those procedures.

These components are at differing stages of development. We are pursuing a breadth-first approach to implementation, where we try to have some version of each of these components before trying to have the "ultimately" correct version of any of them.

In terms of the above components, we have been building the following programs:

Programwriter, which takes a declarative specification of simple programs which need to be written and designs, optimizes, and codes them. The scope of its capabilities includes programs to maintain bank balances and sell tickets for scheduled events [Long 1977].

Susie Software, which is another automatic programmer, for writing manipulation programs for the blocks world. It is a research environment for developing a discourse model which lets Susie engage the user in a dialog concerning the program it is trying to write [Brown 1977].

Proctor, a program which helps a business manager to design a procurement system. It is an "unstructured" questionnaire which provides a framework for a manager to think about his system requirements [Bosyj 1976].

A Digitalis Therapy Advisor, which makes clinical judgments about the condition of a patient who is receiving the drug digitalis, makes further therapeutic recommendations, and can interactively explain its reasoning steps to the user [Swartout 1977].

A question answering system for a relatively simple data base.

We will give an overview of LMS, the theory of grammar, and the interpreter, and discuss other modules as they relate to those central components.

II. The Linguistic Memory System

The OWL LMS is a semantic network with a single primary data type, the *concept*, and a secondary data type, the *symbol*. Symbols are merely strings of characters which denote senses of English words and affixes and have no innate significance. Concepts represent the meanings of all words, phrases, clauses, sentences, etc. of English as well as any needed non-linguistic entities. It is very important to note that, whereas in a traditional semantic network each node of the network represents a single word or item, in LMS each node represents any of the higher-level constructions mentioned above. Thus, where a typical semantic net would identify the meaning of a sentence as some subnet of the whole network, LMS identifies it as a single node of the network.

II.A The Essential Structure of Concepts

Concepts, the nodes of LMS, *have structure*. In fact, we will concentrate on the essential structure of a concept as the primary organizational facility of LMS.

Every concept is defined by a pair, (*genus specializer*), the *essence* of that concept. The genus is another concept, and the specializer is either a concept or a symbol. The genus specifies the general type of the concept; if the genus of concept C is B (i.e., if C = (B *specializer*)), then we imply that C *is-a* B, or C *is a kind of* B.[2] C is called a

---

[2] The general implication of *is-a* or *is a kind of* (*AKO*) links is that "something" (properties, features, place of classification, ways to treat, etc.) is *inherited* by C from B. We will define this more precisely later.

specialization of B, and B is called a *generalization* of C. The specializer serves to distinguish this concept from all other concepts with the same genus; it does not by itself define the concept.[3] The genus and the specializer together identify a concept.

We want to interpret all the concepts in LMS as forming a single taxonomy or tree-like classification system in which the genus points "up" in the taxonomy. To do so, we must designate a single concept, SUMMUM-GENUS, whose genus is itself. That condition makes SUMMUM-GENUS the root of the tree. Further, we insist that no loops may occur in the expression of concepts in terms of themselves or each other (with the above exception for SUMMUM-GENUS). Then, all concepts will form a tree structured classification: starting from any concept in the conceptual memory and successively moving to its genus will always lead to the root concept SUMMUM-GENUS in a finite number of steps. That number will be called the *genus depth* of the concept. We also introduce a notational convenience. So far, we have only allowed a concept to be written as (*genus specializer*). But clearly, the depth of parenthesization for writing any concept will be at least its genus depth, and this is terribly inconvenient. Thus, we allow *equivalence declarations*, such as A = (B C), which allows any appearance of A to stand for an appearance of (B C). A is called the *label* of (B C).

The notion of *derivative subclassification* [Hawkinson 1975] complicates this picture somewhat. It assures that all specializations of a concept are classified the same way the specializers themselves are classified in the conceptual memory. For example, if in the taxonomy both DOG and PIG have genus ANIMAL, then we classify (TAIL DOG) and (TAIL PIG) under (TAIL ANIMAL). The *generalizer* of a concept (A B) is the most specific specialization of A whose specializer is a generalization of B, or, if there are none of these, just A itself.[4] The genus of a concept is thus always either its generalizer or the generalizer of its generalizer, etc. By moving along the successive generalizers from any concept, we must finally reach SUMMUM-GENUS, and the number of steps required is called the *generalizer depth* of the concept.

---

[3] For example, we may represent "dog house" as (HOUSE DOG) and "dog tail" as (TAIL DOG), and although both concepts are specialized by DOG, they are clearly different.

We have now described some of the essential structure of each concept, thus each node, of a conceptual memory. Before we turn to arguing for the utility of this structure to represent knowledge, let us see what the essential structure of the nodes already implies for the semantic network as a whole. In our current implementation, every concept is directly linked to its generalizer and specializer. Every concept is not, however, linked directly to its genus, since the genus can easily be computed from generalizer and specializer links. A typical, but very small, conceptual memory taxonomy is shown in Figure 1.

II.B Attachment

In the previous section, we presented the essential structure of a concept in LMS. The act of creating a new node in LMS is called *specialization*, and we say that we

---

[4] An intermediate concept in the taxonomy, such as (TAIL ANIMAL) in our example, is automatically created by LMS whenever more than one concept may be classified under it. Thus, the generalizer of a concept, and hence the number of times that we need to move from a concept to its generalizer in order to reach its genus, will depend dynamically on what other concepts are in the taxonomy.

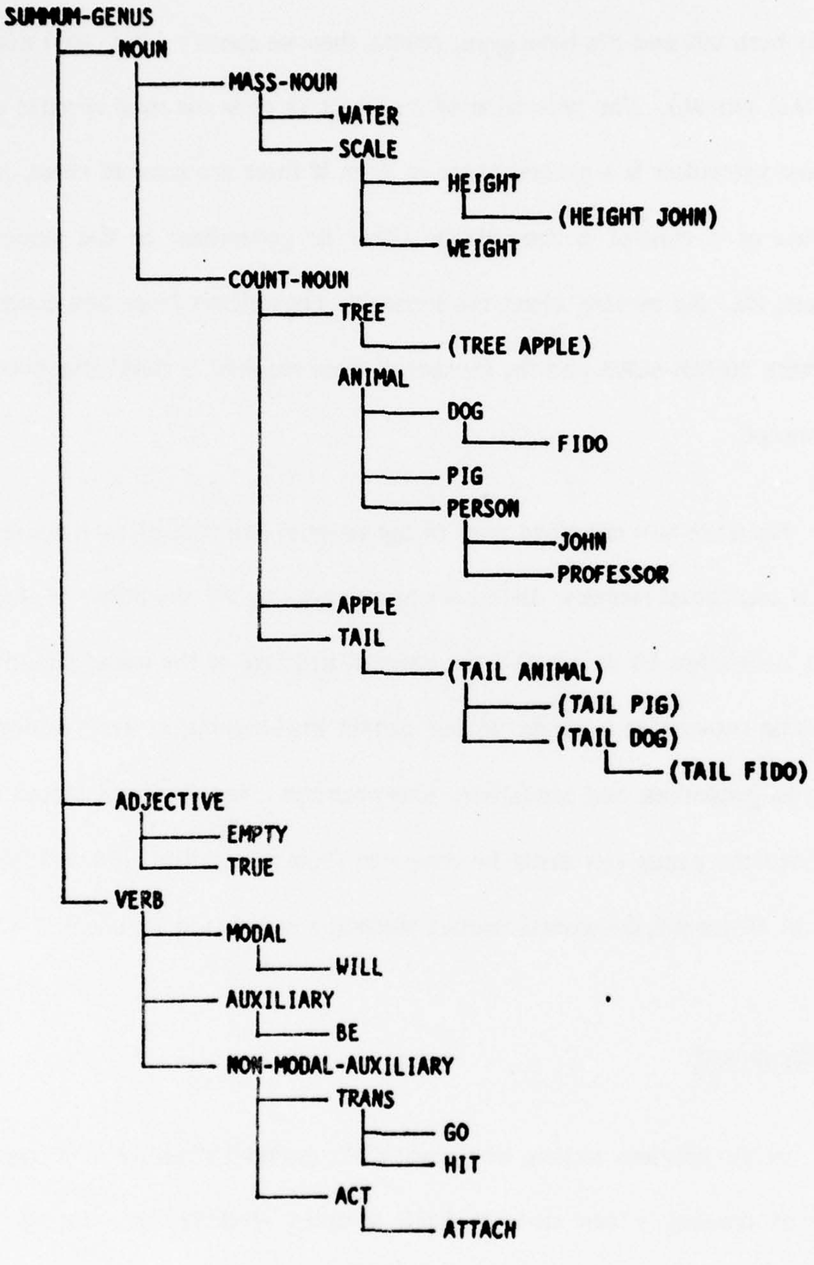
SUMMUM-GENUS
NOUN
MASS-NOUN
WATER
SCALE
HEIGHT
(HEIGHT JOHN)
WEIGHT
COUNT-NOUN
TREE
(TREE APPLE)
ANIMAL
DOG
FIDO
PIG
PERSON
JOHN
PROFESSOR
APPLE
TAIL
(TAIL ANIMAL)
(TAIL PIG)
(TAIL DOG)
(TAIL FIDO)
ADJECTIVE
EMPTY
TRUE
VERB
MODAL
WILL
AUXILIARY
BE
NON-MODAL-AUXILIARY
TRANS
GO
HIT
ACT
ATTACH

Figure 1. A Sample Conceptual Memory Taxonomy

This figure shows the classification of some of the concepts used in this paper into a small conceptual memory taxonomy. The taxonomy is a tree which is shown in the figure by successively indenting branches, as in an outline. Note that derivative subclassification causes the subtree under TAIL to be organized in a similar way to the subtree under COUNT-NOUN. This sample is of course very small and sparse; the taxonomy we currently use has nearly three thousand concepts and a correspondingly more complex organization.

specialize a genus, G, by a specializer, S, to form the concept (G S). As we shall argue, any phrase of English can be suitably encoded as a single concept (though of course it may be a very complex one). When we wish to *reason* with a concept, however, we will find it convenient to introduce an epistemologically distinct *meta-level* representation. For example, if the concept C encodes the sentence "John Smith is a good man" and we wish to represent our belief that C is true, we cannot merely encode with D that "That John Smith is a good man is true," because now the question of D's truth is open for discussion.[5] We retreat to a formal meta-level to make statements about elements of our universe of discourse which are to be taken at face value rather than be subject to interpretation. With such an ability for meta-level description, we see that if C is marked as TRUE at the meta-level, then that is a stronger statement than D. From the former, the Interpreter may conclude C's truth absolutely, while from the latter, only conditionally on D's truth.

The act of *attachment* creates a directed *link* in LMS between two nodes. We write [A B] and say that B is attached to A. Unlike specialization, attachment creates no new concepts. It merely establishes an (unlabelled) link from A to B. The meaning of that connection will depend completely on what A and B are and on whatever is interpreting the connection. We give a few illustrative examples of attachment here:

All concepts B whose generalizers are the concept A are automatically attached to A and are called its *indexed branches* because they are classified directly under A in the specialization taxonomy.

---

[5] It is not merely the representation of truth that is at issue here. A similar treatment is necessary for supposition, hypothesis, "possible futures," and in fact all the fundamental knowledge on the basis of which OWL operates. Of course the effect of the meta-level statements that we allow could alternatively be introduced by suitable conventions for the Interpreter. For example, we could adopt the convention that any statement about which no qualifying information is known is true. We prefer, however, to make such a convention part of the Interpreter and not part of the semantics of LMS.

> Some concepts `(C A)` are attached to the concept `A` and are called its *indexed aspects*. For example, `(AGE JOHN)` may be attached to `JOHN` and encodes `JOHN`'s `AGE` aspect.

Note that both of the above forms of attachment are easily recognizable because the concept to which attachment is made appears as the generalizer or specializer, respectively, of the attached concept. They derive from the essential structure of concepts and serve much the same purpose for the conceptual memory as do index entries in a book's index. These attachments do not really bear information; they are established when the taxonomy is built and are not subject to interpretation or change. Thus, the use of attachment, a meta-level operation, is appropriate.

> Values may be specified by attachment: e.g., `[(AGE JOHN) 49]`.
> Attributes or descriptors may also be attached: e.g., `[JOHN MIDDLE-AGED]`,
> `[(AGE JOHN) (EQUAL (TO (AGE MARY)))]`
> Characterizations may also be specified by attachment: e.g., `[JOHN PROFESSOR]`.

This second set exemplifies storage of information (facts in the object domain), yet we are representing such information at the meta level. This is because we intend that reasoning be based on these facts without further verification. We are willing to guarantee their truth in this domain of application.

III. How English Phrases are Represented as Concepts

In this section, we shall first argue informally that the combination of concepts through specialization provides a mechanism capable of representing English phrases. We will then extend our notion of specialization to deal more rigorously with some problems we encounter.

III.A What Does an English Expression Say?

We view English phrases as expressions built up by combination. To explore what forms of combination are necessary, we examine some *modes of communication* in English and see how they are achieved by combining words and phrases.

1) **Designating.** We use a conventional name for a concept which the listener may be assumed to know. In its simplest form, the conventional name is a word of English, e.g., "apple," which we represent in OWL by `APPLE = (FRUIT "APPLE")`.[6] But we need many more conventional names than we have words in our language. So, we permit the formation of conventional names as combinations (pairs). One member of the pair indicates the class of the concept, the other provides a distinguishing, or specializing, element to make the pair unique. For example, "apple tree" is a conventional name formed by specialization. In LMS, we represent it as `(TREE APPLE)`. Note that no strong distinction is made between conventional names that are compound words and those that are phrases in English. Compare "fire hydrant," `(HYDRANT FIRE)`, and "fireman", `(MAN FIRE)`.

2) **Identifying.** We identify an unnamed concept by combining its class and some (restricting) modifiers. For example, "tall tree," `(TREE TALL)`, and "the apple tree in my yard", `(((TREE APPLE) THE) (IN (YARD MY)))`.[7] The difference between identifying and

---

[6] `"APPLE"` is the LMS notation for the symbol "apple". The concept `(FRUIT "APPLE")` is LMS's notation for the English concept apple.

[7] Some linguists might feel that this phrase should have a different structure, such as `(((TREE APPLE) (IN (YARD MY))) THE)`. We do not claim to have the final answer to all such structural questions, but our formalism allows us to capitalize on whatever insights linguists may have. Where structures are in dispute, we have chosen what seems best to us.

designating is often slight. In designating, we assume that the hearer already knows the concept, whereas in identifying, we ask him to come to know it from what he knows of its components and whatever else we may later tell him. Thus, a "shoe tree," which we might initially accept as an identifying compound without a conventional designation, may come to designate a concept if shoe trees become a popular consumer item. Just as compound words develop from conventional names that are phrases, the latter develop from identifying phrases.

3) **Specifying a grammatical or interpretive aspect**. Chiefly by word affixes, English marks phrases and gives clues to their use in forming sentences and to their proper interpretation. For example, for "books," (BOOK -S), the -S is a grammatical marking for plural on the base concept BOOK. In "hitting," (HIT -ING), and "to jump," (JUMP TO), the -ING and TO play a similar role. This form of marking is called *inflection*. In LMS, inflection is expressed by specializing the concept to be inflected by the affix (or other marker).

4) **Specifying a semantic aspect**. We also represent semantic aspects by specialization. For example, "size of apple," (SIZE APPLE).

5) **Predication**. When we want to say something about an object or action in a factual or hypothetical context, we use predication. Jespersen [1933] calls this *nexus*:

> If we compare *the red door* and *the barking dog*, on the one hand (junction) and on the other *the door is red* and *the dog barks* or *the dog is barking* (nexus), we find that the former kind is more rigid or stiff, and the latter more pliable; There is, as it were, more life in it. A junction is like a picture, a nexus is like a drama or process. In a nexus something new is added to the conception contained in the primary: the difference between that and a junction is seen clearly by comparing, *e.g.*

The blue dress is the oldest.
The oldest dress is blue.
A dancing woman charms.
A charming woman dances.

In our terms, a junction identifies or designates. A nexus, or predication, makes a statement and depends on interpretation for its meaning.

In LMS, we introduce a new notation to express predication: *subject / predicate*. For example, Jespersen's sentence "the oldest dress is blue" becomes `((DRESS OLDEST) THE) / BLUE`. For uniformity of representation and implementational convenience, however, we will implement predication in LMS using specialization by adopting the following convention: The predication `A / B` will be implemented as `((B NEXUS.) A)`.

6) **Itemization**. To specify a group of things related in some simple way, we itemize them. Particular types of itemization are: sequences, conjunctions, disjunctions, sums, products, contrasting pairs, etc. For example "red, white and blue," "3+5+9," and "input/output" are all itemizations. LMS introduces an external notation for such itemizations but implements them by a conventional use of specialization and attachment. The details are unimportant and will not be pursued here. We should add, however, that we feel the notion of sequence to be fundamental.

7) **Naming**. This important mechanism of English will play a major role in our representation formalism. Language often uses context to say concisely what might

otherwise require a verbose specification. In particular, we often use part of a compound to name the whole: `GENERAL` for `(OFFICER GENERAL)`, `CAPITAL` for `(LETTER CAPITAL)`, and `EMPTY` for `(CONTAINER EMPTY)`. In each of these cases, the specializer in context names the whole concept. We shall encounter more general uses of naming below.

III.B Kinds of Specialization

Our treatment of specialization as outlined above is inadequate for some subtler issues of representation. Although we have identified several uses of compound formation in English communication, we have represented them all by the same specialization operation. We form, in a completely similar manner, compound phrases like "the dog," `(DOG THE)`, "sheep dog," `(DOG SHEEP)`, "small dog," `(DOG SMALL)`, and "dog in the yard," `(DOG (IN (YARD THE)))`. For these examples, no problems arise because we can recapture from the specializer itself what kind of compound we have formed. But that will not generally be the case, as we shall see below. In this section, we introduce seven distinct kinds of specialization to enrich our representation scheme.

The English phrase "fat man" is ambiguous. In its common meaning, it stands for a man who is overweight to some degree. The same phrase, however, also describes a professional circus performer of great girth, with whom we associate characteristic forms of dress, behavior, etc. In terms of the modes of communication listed above, we are either (1) designating the circus performer by his conventional name or (2) identifying the man who is overweight by his genus and a distinguishing characteristic.[8] OWL is unique in that we make a *procedural* distinction between these two senses of "fat man." In the first case, "fat" is combined with "man" to identify a pattern in memory, and then that pattern is used to find the referent. In the second case, "man" alone is used to find a pattern in memory, and then items which match this pattern are further checked to see if they pass the pattern designated by "fat." We could imagine a skinny fat man only in the first sense, as referring to the circus performer. But our representational scheme, as presented so far, offers only (MAN FAT) for "fat man," and fails to distinguish the two senses we have discussed.

To preserve the desired distinction between these readings of "fat man," we will mark every specialization with its *meta-type*, which indicates the relation between the concept and its genus.[9] We will represent our overweight man by a restrictive specialization, (MAN*R FAT). A *restriction* (A*R B) may always be paraphrased as "an A which is B," e.g., "a man who is fat," and a restriction always represents a concept which is a kind of its genus with the additional attribute which is its specializer. Note that a tall fat man, ((MAN*R FAT)*R TALL) is not the same as a fat tall man, ((MAN*R TALL)*R FAT), either in real life or in conceptual memory. In a *stereotype*, (A*T B), the specializer has some close relation to the genus but is not necessarily a property of it. Consider not just our circus performer, `(MAN*T FAT)`, but also `(HYDRANT*T FIRE)`, where the relation between "fire" and "hydrant" is a complex one: "a hydrant which is a source of water with which one can put out a fire."

---

[8] In spoken language, the compound representing the conventional name is spoken almost as if it were the compound word "fatman." This additional clue is not available to us via written language.

[9] We are introducing a minor inconsistency here, because we change the meaning of "genus" somewhat. By the rules of LMS, the genus of the concept (A*R B) is A*R, yet we will refer here to A, the concept's *linguistic genus*, as its genus.

The *seven* OWL *meta-types* and their notational suffixes are:

`*R`	restriction	`*A`	aspect
`*T`	stereotype	`*X`	inflection
`*S`	species	`*P`	partitive
`*I`	instance		

`(A*S B)` represents a sub*species* of `A`, where `B` is often just a symbol. This represents a Linnaean classification system in which we assume that different subspecies of `A` form mutually exclusive categories. This is a powerful tool for database search. `(A*I B)` represents an *instance* of `A`. Instances, as species, are mutually exclusive.[10] We thus provide a distinction between classes and individuals by distinguishing instances from species.

An *aspect* specialization `(C*A B)` is a kind of its genus `C`, which is closely associated with its specializer `B`. For example, "height of John," `(HEIGHT*A JOHN)` and "John's leg," `(LEG*A JOHN)`. *Aspects also play the traditional role of programming language* variables. For example, if we have a recipe for pancakes which calls for one egg, that egg will be represented by `(EGG*A (RECIPE*T PANCAKE))`.

---

[10] Some systems further divide instances into manifestations: e.g., "the young Churchill." We would handle this as `(CHURCHILL*R YOUNG)`, where `CHURCHILL = (MAN*I "CHURCHILL")`.

An *inflection*, (A*X B), is used to specify a grammatical or interpretive aspect. It has the unusual behavior that it inherits properties not only from its genus, as all other specialization types do, but also from its specializer. In fact, properties inherited from the specializer override any inherited from the genus. For example, "books," (BOOK*X -S), is plural even though BOOK is singular, because -S carries the plural property.

The *partitive*, (A*P B), is like a semantic version of inflection. The partitive inherits properties from both its genus and specializer, where context determines the appropriate interpretation. Thus, one may first open and then eat a can of beans, first opening the can and then eating the beans.

The above is a short sketch of our approach to representation. A much more complete treatment will be found in [Martin 1977].

III.C Parsing

To translate from strings of English words to their representation, we use an augmented transition network parser based on [Woods 1970]. The OWL parser uses no registers but maintains a constituent stack of concepts with each phrase for which a transition network (TN) is being followed. On every arc is an OWL concept which must be matched for that transition to apply and a set of *combining functions* which manipulate the matching concept and constituent stack.

It is the task of the combining functions to compose OWL concepts representing parts of a phrase into the concept representing the whole phrase. The role of the TN is to invoke the combining functions in the appropriate sequence. The parser operates nondeterministically (via backtracking). Failure leading to backup may occur either because the input string fails to meet word-order constraints (i.e., no match can be found for any arc from a non-terminal node of a TN) or because a combining function rejects a proposed phrase. The conceptual memory contains (expressed via attachment) strictly enforced constraints on case slots of all grammatical concepts. Using these constraints, the combining functions control all compositions such as adjectival and adverbial modification and case assignment for verb phrases. The word-order constraints of the TN's plus the concept-formation constraints in the conceptual memory (as they are used by the combining functions) thus express our grammar.

Two mechanisms of special interest should be mentioned: the use of naming to postpone the introduction of ambiguity, and bidding. Because many English words and phrases have alternate interpretations in LMS (e.g., our "fat man" example), if we were to split our computation nondeterministically every time alternative interpretations of a phrase were available, we would spend a lot of processing effort carrying all those interpretations along until all but one could be eliminated. Further, if more than one interpretation succeeded and the sentence parsed ambiguously, we would have a difficult task localizing the cause of the ambiguity. To avoid these problems, we take a "wait and see" approach[11] and try not to choose the appropriate interpretation until some further constraint forces that choice. Postponing the choice is accomplished by use of the naming mechanism introduced above. In our "fat man" example, we say that conventionally we will form the restriction (MAN*R FAT) as the interpretation of the phrase and we will have in the knowledge base an indication that (MAN*R FAT) names (MAN*T FAT). In this case, the distinction may never have to be drawn during parsing, since no grammatical decisions will depend on it, and it will be some later step of reasoning in the system that may have to choose the "circus performer" interpretation.

---

[11] This technique is motivated by [Waltz 1972] and also applied in a parser by Marcus [1975].

In a typical situation where grammatical distinctions arise early in parsing, we take a slightly different approach from the previous example. The word "drinks" is either the plural of the noun "drink," as in "We had a few drinks," or the third person singular of the verb "drink," as in "Joe drinks beer at dinnertime." Here, rather than choosing one of these as a primary interpretation, we create the neutral (DRINK*X -S) and say that it names both (DRINK*X PLURAL-NOUN) and (DRINK*X THIRD-PERSON-SINGULAR-VERB). To make this scheme work, every combining function must succeed not only when the concepts given to it may be directly combined but also when any concepts named by the given ones may be combined. Matching of concepts on TN arcs is similarly augmented. Further, rules like the above for "drink" generalize, and OWL encodes those generalizations rather than specific naming rules for each concept.[12]

Bidding is another mechanism for deferring a choice among alternatives and avoiding undue nondeterminism. Its application is best seen when considering the attachment of prepositional phrases. For example, in "I rode along the highway in my limousine," we may eliminate "the highway in my limousine" as implausible and attach the prepositional phrase to the predicate (or predication). By contrast, in "I liked the phone in my limousine," the prepositional phrase clearly belongs with "phone." We cannot always make such a definitive judgment: "I saw the man beside our house" places either me or the man beside the house. From further context, the ambiguity may be resolved: "As I approached, I saw the man beside our house." We treat this problem by suspending a path in parsing at a point where it is about to take an arc transition for a prepositional phrase until all possible paths leading to taking such a transition for that same phrase are identified. Then, a conflict-resolving routine is called to permit any number of the possible interpretations to proceed. That routine will, in general, invoke the Interpreter to try to decide which interpretation(s) are best. Its success will depend on the sophistication of world knowledge in the conceptual memory and on the existence of appropriate strategies available to the Interpreter to apply that knowledge. A more specific mechanism which similarly addresses the problem of "selective modifier placement" is presented in [Woods 1973]. We have not yet made any significant use of this bidding strategy.

---

[12] These naming generalizations are called *productive* naming rules. They are applied by the normal inheritance mechanism of LMS, so of course they may be overridden by more specific information in any particular case.

IV. Reasoning

We have implemented an initial version of an *Interpreter* for OWL, which is the basis of the system's ability to reason. It is a large program with many interesting capabilities, of which we will here describe only the central ones. Sunguroff [1976] describes the implementation details of the current version, Brown [1977] is concerned with use of the Interpreter for dialog and the handling of failure, Long [1977] gives another view of the Interpreter's use for automatic programming, and Swartout [1977] discusses the Interpreter's record-keeping and updating capabilities and their relation to explaining program behavior.

So far, we have interpreted OWL concepts as static entities, mere translations of English phrases. The system's action when given the sentence "Prescribe an appropriate dosage of Digitalis for Mr. Jones" cannot be merely to translate that sentence into its internal representation and then stop. But how is it to know what the procedural meaning of some sentence is?

If an OWL concept has a `METHOD` aspect, then it is called a `PLAN` and is something which the Interpreter can *carry out*. When the Interpreter is *called* (its argument is the *call*), it performs the following steps:

1) It tries to match the call to a known plan in the knowledge base. The search for a matching plan proceeds "upward" from the call, so that the most specific plan which matches will be selected.[13]
2) It checks that any required properties on the cases (variables) of the plan occur also on the concepts which will be matched to them.

---

[13] This is a very important idea. With it, we can embed completely specific plans to solve any problems which we know will arise often and will be critical to the system's performance. We also use it to express plans when their choice is dictated not by a reasoned choice but by convention in the application area. If a specific plan is unavailable, slightly more general plans will be attempted, and only if all such plans are found inapplicable will the system resort to some general deductive scheme. We have noted that only when a great majority of specific plans for a domain is available will the system's performance be at an "expert" level. This agrees with our observations that human experts seem to have large portions of their ordinary professional behavior "precompiled" into fixed routines.

3) It creates a new *event*, to record the initiation of execution of the selected plan, and binds all the matched variables.
4) If the plan contains a `PREREQUISITE` aspect, it checks if it is already `TRUE` and if not, then it tries to make it true. This subgoal step of course once again uses the Interpreter.
5) It carries out the steps of the `METHOD`, either in parallel or in sequence, whichever is specified.

We attempt always to use the Interpreter to solve subproblems of an initial problem so that the general matching and reasoning resources we build up will be available at all levels. For example, if `X` is a prerequisite which is not yet satisfied, we merely call the Interpreter with the call `(GET*T X)`. Classical goal-directed behavior can be achieved by use of the `PRINCIPAL-RESULT` case on a plan, which identifies the teleological goal of the plan. Then, if a `GET` is unable to find a plan by its upward search of the concept tree, it may search for a matching principal result and select the plan which promises that result. One other important aspect of the Interpreter is that after every step of interpretation, it dispatches to its next step through the main top-level loop. There, failure-handling and advice-giving procedures may always be invoked to redirect the course of computation by "backing off" from unproductive lines (if they can be recognized).

We are continuing to refine our understanding of the representation of English phrases in the formal notation of OWL and the use of a complex Interpreter which works within that formalism to perform all reasoning tasks which arise in language processing and various application areas.

BIBLIOGRAPHY

[Bosyj 1976] Bosyj, Michael, *A Program for the Design of Procurement Systems*, Cambridge, Mass.: MIT Laboratory for Computer Science TR-160

[Brown 1977] Brown, Gretchen P., "A System to Process Dialogue: A Progress Report," MIT Laboratory for Computer Science TM-79

[Hawkinson 1975] Hawkinson, Lowell B., "The Representation of Concepts in OWL," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*

[Jespersen 1933] Jespersen, Otto, *Essentials of English Grammar*, University of Alabama Press, 1964

[Long 1977] Long, William J., "A Program Writer," Ph. D. Dissertation, MIT (in preparation)

[Marcus 1975] Marcus, Mitchell, "Diagnosis as a Notion of Grammar", in Shank, Nash-Webber, eds., *Preprints from the Workshop in Theoretical Issues in Natural Language Understanding*, 1975

[Martin 1977] Martin, William A., "A Theory of English Grammar" (in preparation)

[Rahmstorf 1977] Rahmstorf, G. and M. Ferguson, eds., *Natural Language for Interaction with Data Bases*, Proceedings of an International Workshop held at IIASA Laxenburg, Jan 10-14, 1977. Laxenburg 1977, IIASA.

[Swartout 1977] Swartout, William R., "A Digitalis Therapy Advisor with Explanations," MIT Laboratory for Computer Science TR-176

[Sunguroff 1976] Sunguroff, Alex, "OWL Interpreter Reference Manual," MIT Laboratory for Computer Science, Automatic Programming Group (unpublished internal documentation)

[Woods 1970] Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, 13, 10 (October 1970)

[Woods 1973] Woods, W. A., "An Experimental Parsing System for Transition Network Grammars," in Rustin, R., ed., *Natural Language Processing*, New York: Algorithmics Press.

[Waltz 1972] Waltz, David L., "Generating Semantic Descriptions from Drawings of Scenes with Shadows," MIT Artificial Intelligence Laboratory TR-271

Official Distribution List

Defense Documentation Center
Cameron Station
Alexandria, Va 22314 12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, Va 22217 2 copies

Office of Naval Research
Code 102IP
Arlington, Va 22217 6 copies

Office of Naval Research
Code 200
Arlington, Va 22217 1 copy

Office of Naval Research
Code 455
Arlington, Va 22217 1 copy

Office of Naval Research
Code 458
Arlington, Va 22217 1 copy

Office of Naval Research
Branch Office, Boston
495 Summer Street
Boston, Ma 02210 1 copy

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, Il 60605 1 copy

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, Ca 91106 1 copy

New York Area Office
715 Broadway - 5th floor
New York, N. Y. 10003 1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375 6 copies

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D. C. 20380 1 copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, Ca 92152 1 copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation & Mathematics Department
Bethesda, Md 20084 1 copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D. C. 20350 1 copy

Mr. Kin B. Thompson
Technical Director
Information Systems Division (OP-91T)
Office of Chief of Naval Operations
Washington, D. C. 20350 1 copy