International Journal Human-Computer Studies 41, 149-177, 1994, Preprint

DEFAULT INHERITANCE IN AN OBJECT-ORIENTED REPRESENTATION OF LINGUISTIC CATEGORIES[†]

Walter Daelemans
ITK
University of Brabant
P.O.Box 90153, 5000 LE Tilburg,
The Netherlands
Walter.Daelemans@kub.nl

Koenraad De Smedt
Experimental Psychology Unit
Leiden University
P.O.Box 9555, 2300 RB Leiden,
The Netherlands
desmedt@rulfsw.leidenuniv.nl

ABSTRACT

We describe an object-oriented approach to the representation of linguistic knowledge. Rather than devising a dedicated grammar formalism, we explore the use of powerful but domain-independent object-oriented languages. We use default inheritance to organize regular and exceptional behavior of linguistic categories. Examples from our work in the areas of morphology, syntax and the lexicon are provided. Special attention is given to multiple inheritance, which is used for the composition of new categories out of existing ones, and to structured inheritance, which is used to predict, among other things, to which rule domain a word form belongs.

Printed Monday, May 16, 1994

[†] We would like to acknowledge the contributions of Josje de Graaf to the research described in this paper. Part of this research was described earlier in De Smedt and de Graaf (1990) and Daelemans (1990). Thanks are due to anonymous referees of IJMMS for useful comments and suggestions.

1 INTRODUCTION

During the last few decades, research in knowledge representation and research in computational linguistics have been getting closer to each other, but in two different ways. On the one hand, the frame-based and object-oriented knowledge representation languages used in AI have widened their grasp on linguistic knowledge: not only domain knowledge has been 'framed', but also syntax, morphology, phonology, and the lexicon (Daelemans, 1987, 1988; De Smedt, 1984, 1990). On the other hand, dedicated linguistic formalisms have been enriched by ideas coming from established work in knowledge representation. The incorporation of inheritance in unification-based formalisms (e.g. Shieber, 1986) is an example of such an enrichment.

We argue that it is better to refine and further tailor a general (but sophisticated) computer language to the needs of the linguistic domain, than to design a specialized language from scratch. Practical reasons for this preference are, first, that there is no point in reinventing the object-oriented wheel, and second, that a general purpose language is more useful than a special-purpose one. We also wish to consider linguistic competence as a sub-component of knowledge in general as much as possible, thereby allowing maximal generalization. Proponents of Word Grammar (Hudson, 1984; Frazer & Hudson, 1992), among others, have set the same goal. In this context, we argue that the application of AI languages to linguistic knowledge deserves more attention. Specifically, the mechanisms for inheritance, encapsulation, and polymorphism, which are common in frame-based and object-oriented programming are essentially domain-independent. They are eminently suited to the representation of many kinds of knowledge, including linguistic knowledge.

To illustrate our point, we will show several examples of the interplay between inheritance, encapsulation and polymorphism in the representation of linguistic categories. We will concentrate on several aspects of default inheritance in this paper, and we will use selected parts of our work in morphology, syntax and the lexicon of Dutch as an example domain. Due to space limitations, our examples will be concise and will serve only to illustrate the main theoretical concepts. Applications dealing with real world problems such as the generation of written and spoken language in an actual man-machine interface fall outside the scope of this paper.

The remainder of the paper is organized as follows. In the remainder of section 1, we will present our basic starting points with respect to the object-oriented representation of knowledge in general, and linguistic knowledge in particular. In section 2, we will describe the architecture of an object-oriented model of *morphology* (the computation of word forms). In Sections 3 and 4, specific issues concerning inheritance will be discussed and illustrated with examples from this morphology model. Section 5 illustrates the use of inheritance to improve the representation of *feature structures*. Here we will focus on the area of *syntax* (the computation of sentence structure), using a unification grammar as a concrete example. Section

6 discusses an alternative representation for some problematic cases. The final section concludes with evaluative comments and a future outlook.

1.1 Language as an open system

What we do in defining a knowledge representation language is to make an abstraction over a class of representational structures and introduce a syntactic mechanism to express that abstraction. The resulting new primitives will manage the complexity of knowledge so that programs will be more understandable, modularity will improve, and efficiency will be gained in several other ways. But the relevance of postulating new representational primitives goes beyond mere productivity concerns. They state generalizations about the representational structures used by processes in intelligent systems (Steels, 1978). As with all empirical generalizations, it may not be possible to absolutely prove that it is adequate or valid, but it may be possible to find cases demonstrating why a generalization is notationally adequate, for example from a viewpoint of economy.

Default inheritance in taxonomies of object classes or types is such a representational primitive. It refers to the ability to generalize and specialize mental concepts, and to reason by virtue of prototypes while keeping the system open to exceptions. This form of reasoning pervades much of our common sense knowledge but is also a high-level mechanism for the symbolic manipulation of important concepts in a scientific domain. Fikes and Kehler (1985) point out that representations based on hierarchically ordered, structured objects "capture the way experts typically think about much of their knowledge, provide a concise structural representation of useful relations, and support a concise definition-by-specialization technique that is easy for most domain experts to use." This is also true for the way linguists generally think.

The software engineering advantages in using object-oriented programming (modularity, conciseness, reusability) are well-documented and apply *a fortiori* to the design and implementation of natural language processing systems, where hierarchical reasoning with defaults is necessary for a practical and realistic representation of linguistic knowledge. A natural language is an open system. The development of extensible and adaptable natural language processing systems crucially depends on a knowledge representation paradigm within which generalizations are effectively exploited (Jacobs, 1985).

This is not to say that inheritance is only a software engineering tool. Especially in a lexicalized grammar, the avoidance of redundancy is mandatory, while the possibility to incorporate exceptions must be left open (Flickinger, 1987). We are trying to model linguistic knowledge the way linguists typically work. Many grammars, especially traditional ones, are implicitly organized by means of abstraction over linguistic categories. The hierarchical relations between grammar concepts may emerge in the organization of a grammar book: one finds a chapter on *The noun*, which in turn has a specialized section on *The proper noun*, etc. While we think this is basically the right approach, it is nonetheless necessary to make the nature of inheritance in the hierarchy much more explicit. This can be achieved by developing

formal theories of default inheritance and proposing algorithms for their implementation. In short, the achievement of a hierarchical theory of language must be supported by a knowledge representation framework which incorporates primitives for hierarchical reasoning. For a broader thematic and historical overview of the use of inheritance in linguistics and natural language processing, we refer to Daelemans et al. (1992).

1.2 Inheritance as a multi-purpose mechanism

An important feature of object-oriented languages is that they provide some kind of mechanism for objects to inherit their structure and behavior from other ones. The application of inheritance can be seen from different points of view which we will list below. Notice that these perspectives are overlapping: *stepwise refinement*, for example, is a particular use of *specialization* .

1 *Specialization.* From a conceptual point of view, inheritance allows the representation of more specific objects as specializations of more general objects. In this way a *specialization hierarchy* is produced, which corresponds closely to the hierarchy of *'is a'* links in a semantic network. For example, the objects *vowel* and *consonant* might be specializations of *phonological segment.* A hierarchy of grammar concepts might contain objects for *word* and its specializations *noun, verb, preposition,* etc.

2 *Combination.* Another kind of inheritance, *multiple* inheritance, represents the behavior of an object as a combination of the behaviors of two or more other objects. This is often done if an object needs to integrate knowledge from different sources or perspectives. For example, the behavior of the objects *plural* and *noun* could be combined to describe a plural noun. Objects are rarely combined on the basis of equality. It will often consist of the addition of a few special features (for example those of *plural*) to a more general object (in this case *noun*). The secondary object whose features are 'added' is sometimes called a *mixin*. A *transitive plural strong verb* could be defined as a combination of the object *verb* with the mixins *transitive, plural,* and *strong.*

3 *Stepwise refinement.* A program can be constructed by first modeling the most general concepts in the application domain, and then dealing with special cases through more specialized objects. The programmer is not so much concerned here with the construction of a taxonomy but uses refinement as a programming methodology. Stepwise refinement by specialization can be compared with the well-known methodology of stepwise refinement by decomposition (Wirth, 1971). It is significant that the effort of defining an object is proportional to the extent in which it differs from other objects. Thus, refinement is not only useful as a programming methodology, but it can also be thought of as a general cognitive mechanism, for it reflects a principle of least effort.

4 *Avoiding redundancy.* From the point of view of data storage, inheritance is a way of knowledge sharing. A piece of information which is necessary in many objects needs to be stored in only one object, where others can access it. Avoiding redundancy in this way reduces memory requirements. It also improves modularity, because the shared knowledge only needs

to be updated in one place. Again, the principle of system economy is thought of as a general cognitive principle and not just a software engineering strategy.

5 *Predictions about new objects*. When new objects are created in the course of the computation, inheritance can be used to predict their default behavior. For example, if we say that the mother of *person* inherits from *woman*, we predict that if a certain object is the mother of a specific person, it will—at least by default—be a woman. *Structured* inheritance is a mechanism capable of making such default predictions. It offers many opportunities for the representation of linguistic knowledge and will be dealt with in more detail below.

It is important to mention here that our view of inheritance is always based on *defaults* rather than absolute and irretractable statements. If we say "birds fly", then we mean "birds typically fly" and we have no problems to accept and handle exceptions—for instance, ostriches and penguins—adequately. Similarly, if we say "nouns are countable", then there may still be a special class of uncountable nouns. All inherited knowledge only holds in so far as it is not overruled by knowledge in the inheriting object. Consequently, our view of inheritance incorporates an *implicit blocking* of defaults in view of more specific knowledge.

We conduct our work in the frame-inspired object-oriented languages CommonORBIT (De Smedt, 1987, 1989) and KRS (Van Marcke, 1987) which are based on the notion of *prototypes*. For examples we will use the syntax of CommonORBIT, which is implemented as an extension of Common LISP. Whereas in some other object-oriented languages, every object must belong to some class (or type, or flavor), this is not the case in CommonORBIT. Instead, an object in CommonORBIT can be a prototype (or model, or proxy) for any other object. This untyped view of inheritance uses 'is-like' (or 'shares-with', or 'delegates-to') relations, which have a more general semantics than the 'is-a' relations in typed object-oriented languages. A similar view of inheritance based on prototypes and delegation has been proposed by Lieberman (1986).

2 A HIERARCHICAL MODEL OF MORPHOLOGY

We have developed and implemented a model of Dutch morphology which generates structured, phonologically and orthographically specified word forms from their bases in the lexicon. The model, which covers Dutch nouns, verbs and adjectives, consists of the following modules: (1) a hierarchical lexicon with objects representing simple unstructured words (base forms), (2) a morphological component consisting of a hierarchy of morphological categories and associated rules, and (3) a phonological component consisting of a hierarchy of phonological categories and rules. The three modules are not independent in their representation; as will be shown below, they are only different locations in the topology of a single large lexical hierarchy of linguistic concepts. The model is meant to be embedded in a

larger system for natural language generation[1] for which other components are under development. The input to the morphology component consists of base forms with grammatical features such as *singular*, *third person*, etc. On the output side of the morphological component, structured word forms are passed on to a phonological component, which applies a set of phonological rules (for instance, *assimilation*) to yield fully specified phonetic strings, and a spelling component which produces orthographic strings. In addition, the morphological component may be used to extend the lexicon with newly formed words. The system is roughly outlined in Figure 1.

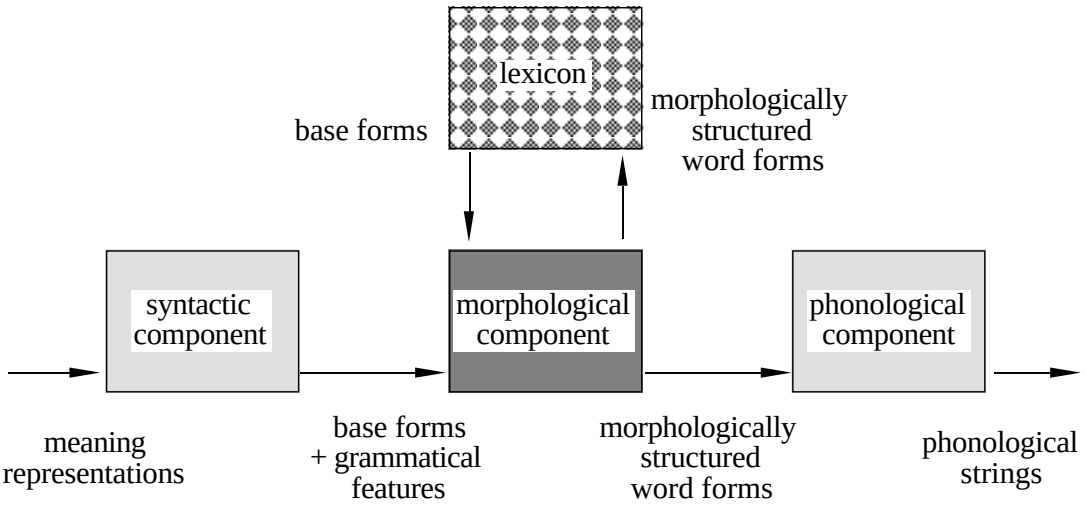

Figure 1

The place of the morphology component in the natural language generation system.

2.1 The lexicon as a hierarchy

The morphological knowledge needed for the computation of word forms is embedded in a larger object-oriented lexicon architecture in which maximal use is made of the information combination, default reasoning, and sharing possibilities of multiple default inheritance. Linguistic knowledge is organized in a specialization hierarchy of objects representing semantic, syntactic and morphological categories. Lexical base forms are the bottom nodes of this hierarchy. They have associated with them idiosyncratic information (their lexical representation), and inherit from different objects in different subhierarchies (semantic, morphological, syntactic). Both procedural knowledge (i.e. rules) and declarative knowledge (for example features) become available to the base forms by default inheritance.

---

[1] One may wonder how suitable these methods are for applications other than generation. We will not address this question in depth here, but we admit that it is not straightforward to rearrange the system so that it is suitable for parsing. Although generation and parsing are assumed to operate largely on the same kinds of structures, it is not possible to simply run the programs described here 'in reverse'.

Part of the lexical hierarchy for English is shown in Figure 2, where relations allowing inheritance are depicted as arrows. The base form *(to) pay* is defined as inheriting from *transitive, regular verb,* and *commercial action.* The hierarchy represents surface semantic, syntactic, and morphological knowledge. Through default inheritance, the base form gains access to the syntactic knowledge that it is a normal *verb* (not an auxiliary), that its *(direct) object* is an noun phrase (NP) with the *accusative* case, and that its *subject* is an NP with the *nominative* case. The semantics inherited includes the knowledge that the *agent* of the action referred to by *pay* is the referent of the *subject*, and the *patient* is the referent of the *object.* By virtue of the inheritance link to *regular verb*[2], a number of word formation rules (methods) become available to *pay.* In contrast, *buy* has a link to *irregular verb,* and consequently inherits different morphological rules. These rules allow new word forms to be created and assigned their proper place in the hierarchy. That way, lexical information becomes available through inheritance to these derived forms as well.

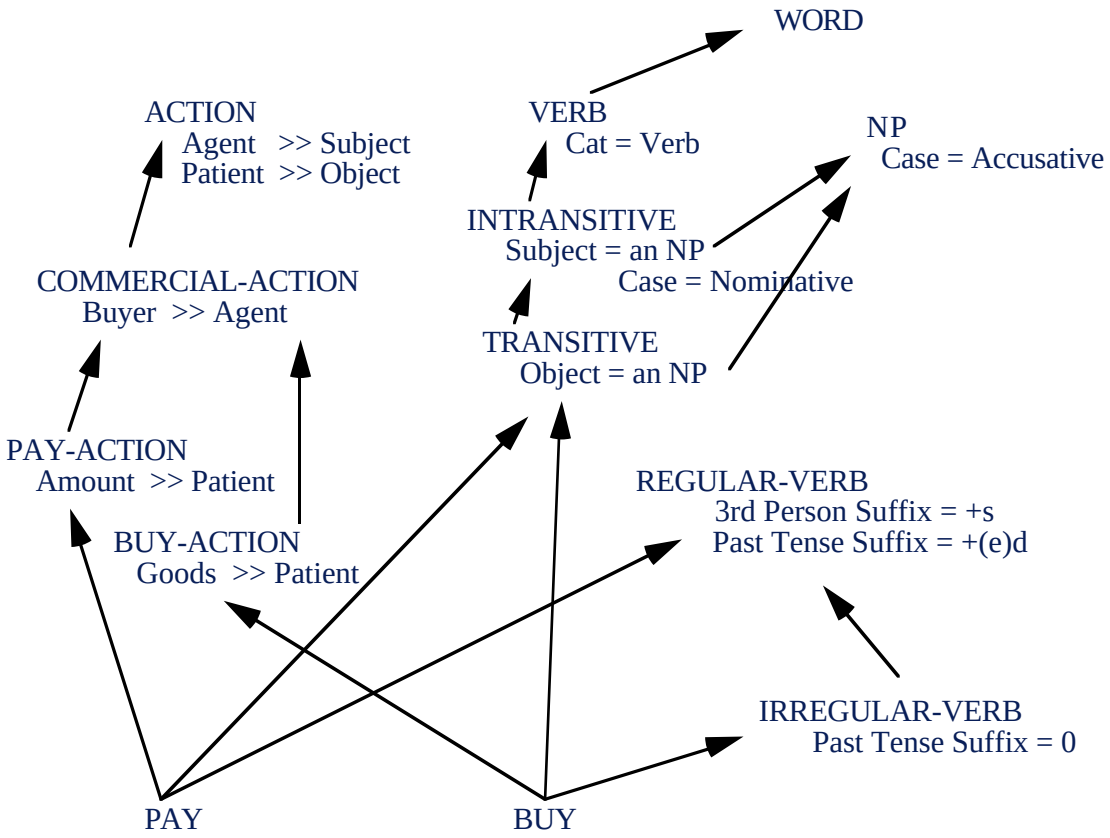

Figure 2

Integration of knowledge in a hierarchical lexicon

The uniform representation of lexical knowledge enables the definition of the interaction between different sources of knowledge at a high level of abstraction (for example, the

[2] Morphologically speaking, *to pay* is seen as a regular verb despite superficial spelling changes in its past tense and past participle.

definition of semantic agents as syntactic subjects), while at the same time allowing for exceptions and sub-regularities (for example, subjects of passives are patients). In the remainder of this article, we will mainly focus on the organization of the morphological subhierarchy within this lexicon architecture.

2.2 Word formation as a recursive process

Dutch morphology is somewhat richer than English. Dutch polymorphemic word forms may be derived from simpler forms by means of prefixing, for instance, *her+doe* (redo), by suffixing, for instance, *groen+ig* (greenish), or by a combination of both prefixing and suffixing, for instance, *ge+werk+t* (worked, past participle). This process covers both derivation and inflection[3]. In addition, words may be compounded, for instance, *lucht+haven* (airport) or can undergo simultaneous compounding and affixing, for instance, *drie+kopp+ig*[4] (three-headed). As can be expected, polymorphemic word forms may themselves be the base for other word formation processes as if they were simple morphemes, which makes word formation a recursive process. It is easy to assign a structure to polymorphemic words which reflects the ordering of the various recursive derivations and compoundings by which they are generated. For example, the Dutch word *ge+her+structur+eer+d+e* (restructured) can be represented as a bracketed structure where each successive recursive layer of the derivation is represented as a concatenation of a prefix, a base, and a suffix, where the prefix or the suffix may possibly be empty, represented by a space (" "):

(" " (ge+ (her+ (" " structur +eer) " ") +d) +e)

This representation, which we call the *lexical representation*, is computed by means of the list concatenation of the lexical representations of a (possibly empty) prefix, a base, and a (possibly empty) suffix. The CommonORBIT code of the prototypical wordform contains a procedure for creating such a list:

```
(DEFOBJECT WORD
  (PREFIX (A 0-MORPHEME))  ;default = empty
  (BASE :IF-NEEDED #'IDENTITY)  ; default = the word itself
  (SUFFIX (A 0-MORPHEME))  ;default = empty
  (LEXICAL-REPRESENTATION :IF-NEEDED
    #'(LAMBDA (SELF)
        (LIST
          (LEXICAL-REPRESENTATION (PREFIX SELF))
          (LEXICAL-REPRESENTATION (BASE SELF))
          (LEXICAL-REPRESENTATION (SUFFIX SELF))))))
```

---

[3] Inflection operates under the influence of grammatical features such as number, person and tense. Derivation covers wider shifts of meaning and may also involve a shift in grammatical category, e.g., the English adjective *computable* is derived from the verb *compute*.

[4] Our morphology abstracts from spelling adjustments, e.g. reduplication of the final consonant in *kopp*. These adjustments are carried out by a separate spelling module which is active on the same level as the phonological component.

```
(DEFOBJECT  0-MORPHEME
  MORPHEME
  (LEXICAL-REPRESENTATION " "))   ;space means empty
```

The first DEFOBJECT form defines an object *word* with aspects *prefix, base, suffix* and *lexical-representation*. The second DEFOBJECT form defines the empty morpheme. Aspects of type :IF-NEEDED contain functions that compute a value only when needed. For more details on the syntax of CommonORBIT, we refer to De Smedt (1987). The lexical representation of a morpheme is a string consisting of phonological segments[5] and boundaries, supplemented by diacritics (accent markers) for stress, etc. The lexical representation of a word is computed as a list structure containing other lexical representations, i.e. those of its prefix, base and suffix. Eventually, a *phononogical* string can be derived from such a list by removing the brackets and concatenating the contained strings. This phonological string still abstracts from regular phonological variation. The ultimate *phonetic* string is determined only after general phonological rules (for instance, assimilation) have operated on it.

2.3 Objects as a uniform representation

Our work uniformly represents all linguistic categories, such as phonemes, morphemes, etc. as structured objects. Several basic tenets of object-oriented programming are relevant to linguistic representation. One of these is the principle of encapsulation, the localization of structure and behavior in the object to which it conceptually belongs. An object can be seen as a collection of properties and (potential) behaviors. Furthermore, the identity of an object is important[6]. This identity can be used, for instance, to relate stems of complex word forms such as *in**herit*** and ***herit**age* (see also Russell, Ballim, Carroll and Warwick-Armstrong, 1992, for a treatment of separable verbs along these lines).

This implies that in our approach to morphology, not strings are the basic units (as in two-level morphology or generative phonology), but structured morphological objects. The phonetic representation, which takes the form of a string, and the lexical representation, which takes the form of a list, are only two of many attributes associated with word form instances. Similarly, string concatenation is only one aspect of morphological processing, which involves object creation and multiple inheritance of properties. This approach makes it easier and more natural to handle discontinuous morphemes, morphological processes that have no effect on word form (for example type conversion), and multi-word lexical entries (idioms).

---

[5] For ease of reading, we will normally use spelling rather than pronunciation in this paper. Nevertheless, the lexical representation contains in principle phonological rather than orthographic units, because some morphological processes may depend on aspects of pronunciation.

[6] For a discussion of the importance of object identity in the representation of linguistic knowledge, we refer to Van der Linden et al. (1988).

Our work follows De Smedt (1984), who accounts for some regularities and exceptions in the inflectional morphology of Dutch verbs by employing default inheritance in a hierarchy[7] as depicted in Figure 3.

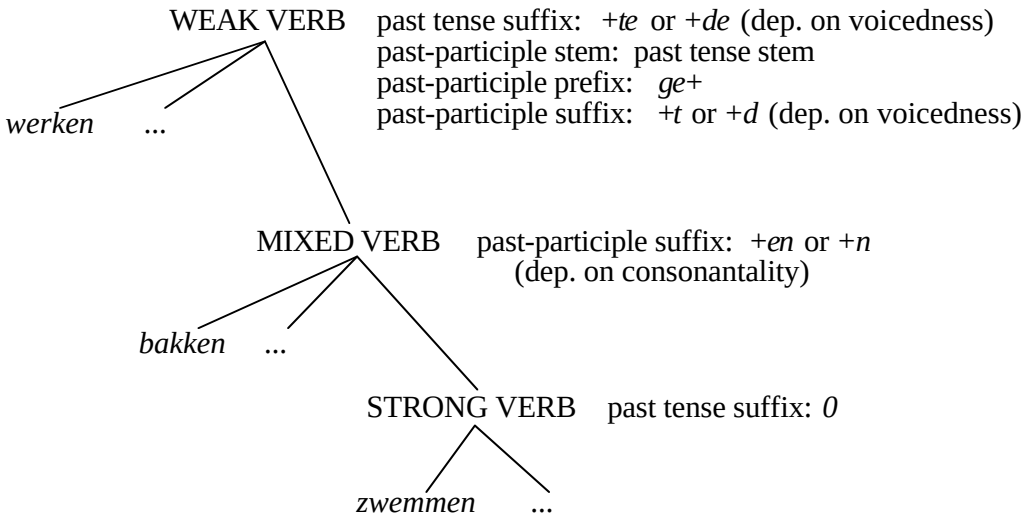

Figure 3

A partial hierarchy of Dutch verbs (adapted from De Smedt, 1984)

The top node of this hierarchy, *weak verb*, is the most regular kind of verb and therefore is the prototype object for all other verbs. Actually, this object inherits from an even more general object, *word*, and so on. The inflectional behavior of the weak verb is represented in a number of aspects which each compute a stem, prefix, and suffix for each of its inflections. All weak verbs have the object *weak verb* as a prototype. For example, *werken* (to work) is an object which inherits its inflectional behavior directly from *weak verb*. There is a specialization representing a prototype, *half-strong verb*, which is partly regular but has an exceptional past participle suffix. This contradicts the specific information associated with its *weak verb* prototype for this particular form, which is thus overruled. Again, there are a number of verbs which inherit their morphological behavior from *half-strong verb*, and so on. This representation provides a non-redundant and generalization-capturing account of Dutch verb inflections.

It is significant that the hierarchies in Figures 2 and 3 contain lexical items as well as morphological categories. Thus, there is no strict separation between the lexicon and the body of morphological knowledge. Instead, there is a smooth transition from the most general categories to the most specific ones—individual words. This is a direct consequence of the uniform representation of linguistic knowledge as objects. A uniform framework has the advantage that it allows for the same general principles to be applied to a variety of knowledge units (Jacobs, 1985).

---

[7] This example is also discussed in Gazdar (1987).

3 MULTIPLE INHERITANCE AS COMPOSITION

Specialization is but one facet of the inheritance mechanism. The other is the definition of new objects as a combination (or composition) of several prototypes. This presupposes the handling of *multiple* inheritance. In this section, we will show how this mechanism can be exploited in at least three ways.

3.1 Combined objects as mixins

One way to exploit combination is the creation of categories that combine information from different knowledge sources. This will often amount to the addition of a few special features to a more general category. For example, a *nominative plural NP* can be seen as an object that inherits from *nominative, plural,* and *NP*. The prototype NP is the most substantial category, whereas *nominative* and *plural* are secondary objects whose features are 'added'; they are sometimes called *mixins*. We will avoid the many thorny issues in multiple inheritance (see for example Touretzky, Horty & Thomason, 1987), but we must nevertheless address the question of how conflicts between contradicting knowledge in the composing objects can be resolved. It is clear that mixins are usually meant to have priority over the defaults in the more general categories. In the case of a *nominative plural NP,* the mixins *nominative* and *plural* have priority over the prototype *NP,* which may be *accusative* and *singular* by default. We will call the relative ordering of composing objects *local* or *definitional precedence.*

However, we must make sure that definitional precedence does not violate hierarchical precedence. For example, suppose that *rondrijden* (to ride about, to tour), a Dutch compound verb, inherits from both *compound* and *verb*. The prototypes compound and verb are defined separately, so that the knowledge encapsulated in them can be reused for different combinations, for example compound noun, compound adjective, etc. Suppose, furthermore, that we specify that the knowledge in *compound* has definitional precedence over that in *verb*. Any knowledge in *compound* will precede that in *verb*. In principle, we could implement this as a depth-first search in the hierarchy (from left to right in Figure 4).

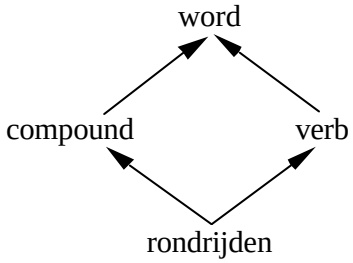

Figure 4

Multiple inheritance with a common prototype

When combining knowledge from different sources, it cannot be excluded that the different sources share a common prototype, for instance, the object *word* in Figure 4. A depth-first search in the hierarchy will respect the priority of *compound* over *verb*, but will not do justice

to the specialization relations. The common prototype must not be considered before more specialized objects in the hierarchy. This can be formulated as the following principle, which has also been advocated by Ducournau and Habib (1987):

> *Specialization vs. Multiplicity:*
>
> Inheritance must follow the specialization partial order; therefore, in any case the specialization relation excels the local (definitional) precedence of prototypes.

Following this principle, both *compound* and *verb* will have precedence over *word* in the example of Figure 4.

3.2 Multiple inheritance as biased semi-compositionality

Seen from a different perspective, the combined operation of the principles of definitional precedence and specialization yields a kind of 'biased' semi-compositionality that is common in descriptions of natural language phenomena. One prototype usually plays the role of *head* while the others play the role of *modifiers* with different priority. Furthermore, the information which is compositionally inherited from the prototypes can be overruled by locally specified information to express subgeneralizations and exceptions.

The use of composition as consisting of a head and modifiers can be exemplified by the semi-compositional nature of compounds such as *blackboard*. This compound could be defined as inheriting from *board* and *black* (in that definitional precedence order), biasing the composition of information toward inheritance of syntactic and morphological features from *board*. In other words, *blackboard* is first and foremost a noun like *board,* and inherits additional knowledge from the adjective *black.* However, the combination is not only biased, but also only semi-compositional, because in this case, semantic information inherited from the semantic representations of the parts may be overruled (blackboards may be green by default).

3.3 New categories as mules

A third use of composition creates new objects whose behavior is 'in between' that of two or more others. Think of such a new object as a mule which is the young of a donkey and a horse. We will give a more elaborate example in the domain of the morphology of verbs, in order to show this use of multiple inheritance. In Dutch, there are in fact more kinds of strong and half-strong verbs than those dealt with in section 2.3. Some half-strong verbs are different in that they have a vowel change in the past participle; this kind will be called *half-strong verb 2.* A third kind is partially weak and partially strong, but in exactly the opposite way: they have a strong past tense (with vowel change), but a weak past participle; let us call this kind *half-strong verb 3*. The new hierarchy is depicted in Figure 5.

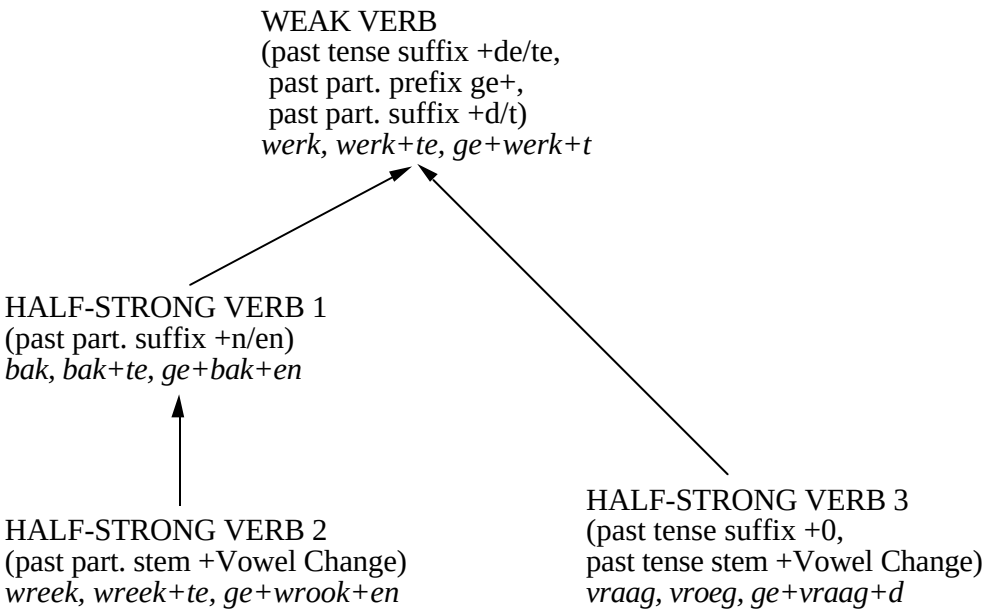

Figure 5

Objects for Dutch half-strong verbs

The strong verbs in Dutch can also be divided into two kinds, one which exhibits a vowel change in the past participle and one which does not. This offers the opportunity to use multiple inheritance for object composition, because the behavior of the strong verbs can be found distributed among the various half-strong verbs of Figure 5. The resulting representation, shown in Figure 6, is especially powerful because the definitions for the strong verbs consist *only* of the combination of the objects they inherit from, *without* any additionally specified knowledge. The multiple inheritance principle (specialization vs. multiplicity) makes sure that more specialized prototypes have priority over more general ones. Thus, for example, *strong verb 1* inherits from both *half-strong verb 1* and from *half-strong verb 3* before the more general knowledge in *weak verb* is considered. Thus, the defaults in *weak verb* are effectively blocked. In this hierarchy, there is no conflict between branches because the main classes of half-strong verbs are opposite and thus complementary; they each provide differing specific information which does not contradict each other. However, the principle of specialization vs. multiplicity is crucial.

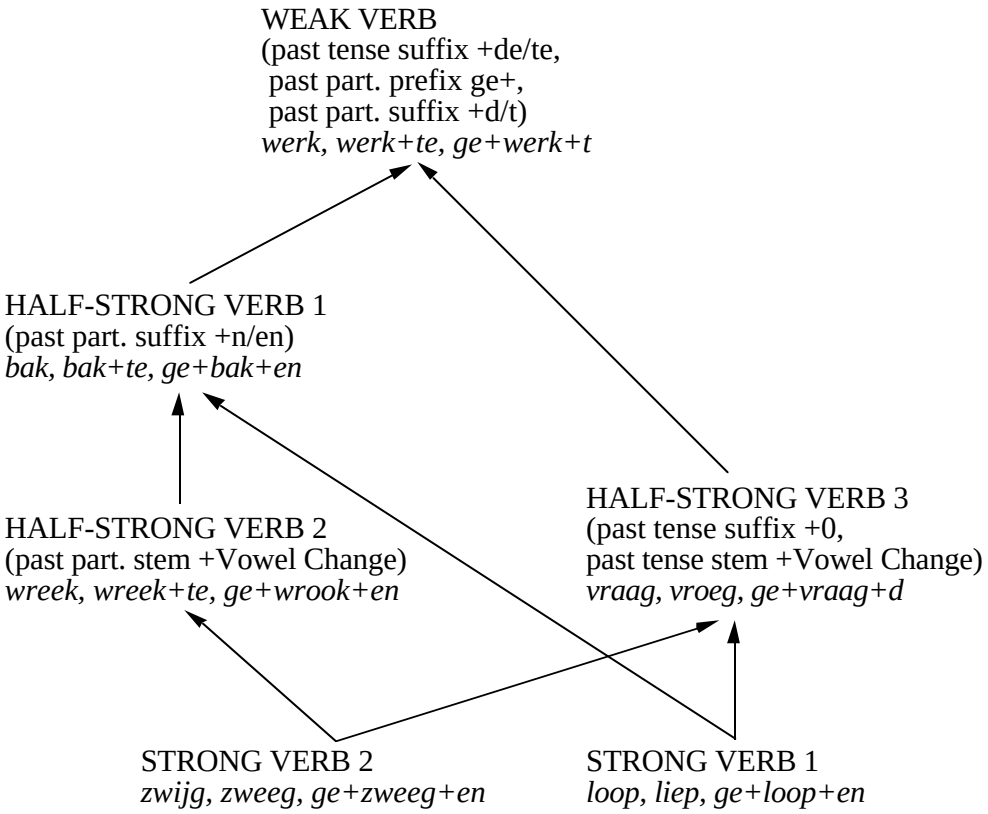

Figure 6

Revised hierarchy of Dutch verbs

4 STRUCTURED INHERITANCE

In this section, we want to devote special attention to the application of *structured* inheritance in morphology, because this inheritance mechanism, though very powerful, has received little attention in the computational linguistics literature. This is also true for the more general OOP community: CommonORBIT and KL-ONE are among the very few object-oriented languages that incorporate this mechanism.

4.1 Structured inheritance as an automatic mechanism

Structured inheritance is a mechanism in frame-based and object-oriented representation which models a slot after one higher in the hierarchy. When an object (or frame) inherits from another one, the fillers of its aspects (or slots) will automatically inherit from corresponding fillers in the higher level object. Structured inheritance is central in the frame-based language KL-ONE (Brachman & Schmolze, 1985) and in CommonORBIT.

Let us first consider a simple example in a non-linguistic domain. Suppose we want to express the knowledge that the mother of a person is normally a woman who is (at least by default) not a virgin. We could represent this by creating an object for the concept *person* and defining a mother aspect which is filled by an object representing the prototypical mother, i.e. a

'non-virgin woman'. The relations allowing inheritance are graphically represented by means of arrows in Figure 7. Notice that there is a stacking of defaults, one in *woman* and one in the mother of *person*. The corresponding code in the object-oriented language CommonORBIT is as follows (where each expression is followed by the result of its evaluation):

```
(DEFOBJECT PERSON
  "The mother of a person is, by default, a woman who is,
   by default, not a virgin."
  (MOTHER (A WOMAN
             (VIRGIN? NIL))))
#<object PERSON>

(DEFOBJECT WOMAN
  "A woman is a person of the female sex."
  PERSON
  (SEX 'FEMALE))
#<object WOMAN>
```

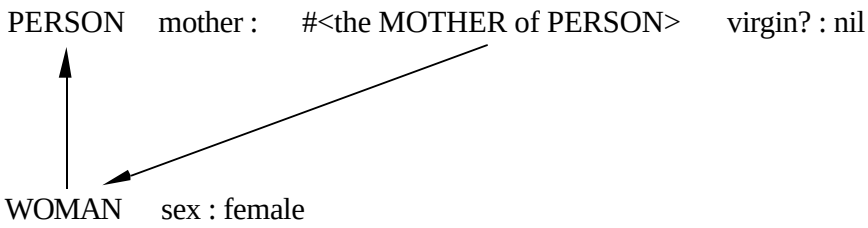

Figure 7

Structured inheritance: a non-linguistic example

Structured inheritance is a way of using this representation to infer some property of a specific mother. When we ask for the mother of a specific person, say Olivia, we obtain an object which inherits from the prototypical mother of *person*. Figure 8 graphically illustrates this inference, which is triggered by the following CommonORBIT code:

```
(DEFOBJECT OLIVIA WOMAN)
#<object OLIVIA>

(MOTHER 'OLIVIA)
#<the MOTHER of OLIVIA>

(SEX (MOTHER 'OLIVIA))
FEMALE

(VIRGIN? (MOTHER 'OLIVIA))
NIL
```

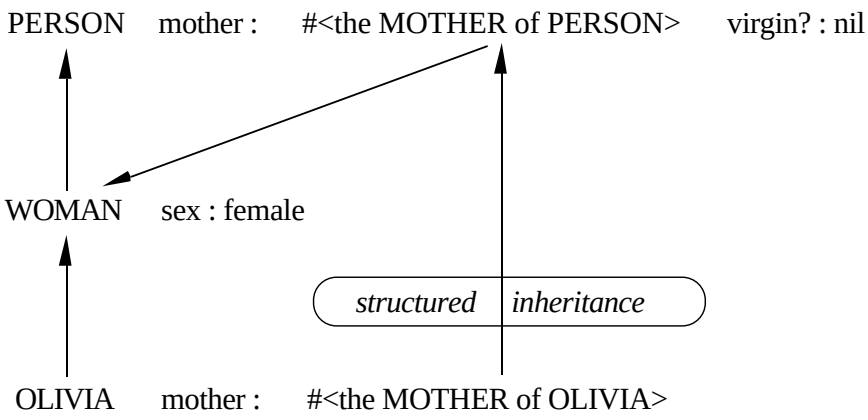

Figure 8

Structured inheritance at work

The identity of the newly made object is important. For each object which inherits from person, a new mother object is created, such that every person object is provided with its own unique mother[8].

Summing up, structured inheritance has been described as the ability to "… preserve a complex set of relations between description parts as one moves down the specialization hierarchy" (Brachman & Schmolze, 1985:177). In systems providing structured inheritance, the inheritance mechanism is not limited to simply sharing or copying a value, but it models an object and the network of all its associated objects after one higher up in the hierarchy, thereby potentially eliminating considerable redundancy. Since structured inheritance is transitive, knowledge from various levels can be combined. Finally, since defaults are handled as in ordinary inheritance, it remains perfectly possible to represent exceptions. For example, it is straightforward to represent an object with a virgin mother in CommonORBIT as follows:

```
(DEFOBJECT JESUS PERSON
   ((MOTHER VIRGIN?) T)
#<object JESUS>
```

4.2 Derivation rules as generic functions

To see how structured inheritance works in the linguistic domain, we first explain how morphological rules are represented. The basis of the morphology model is formed by objects representing *morphosyntactic* categories, that is, categories such as *noun, adjective, verb,* etc.

---

[8] One may put the question, how we can represent the fact that for example two sisters have the same mother. Again, this can be achieved by structured inheritance, in combination with the overruling of a default, to define a sister as a woman with the same mother, as shown in the following code:

```
(DEFOBJECT PERSON
  (SISTER :IF-NEEDED (SELF)
    (A WOMAN (MOTHER :OBJECT (MOTHER SELF)))))
```

which have a syntactic role and a morphological behavior. As is usually done in computational linguistics, the category label is represented as a feature. For example, an *adjective* (ADJ) is a *word* with a *category* feature which has value *adj*. This prototype adjective is defined in CommonORBIT as follows:

```
(DEFOBJECT ADJ
  WORD
  (CATEGORY 'ADJ))
#<object ADJ>
```

Word formation is characterized as the computation of a derived form (inflection, derivation)[9] from a base form. Each base form belongs to one or more classes, each corresponding to part of the domain of a derivation rule. The rule itself is represented as a *generic function* whose behavior depends on the class of the objects in its domain. Hence, the conditional part of a rule can be *distributed* over a number of categories.

Morphological rules are attached to the objects in their domain, i.e. they are defined as procedural aspects (or methods) of objects representing morphological categories. By way of example, the rule generating a derivation with the suffix *+ig* ('ish'), for example *groen+ig* ('greenish') is attached to a category representing the domain of this rule:

```
(DEFOBJECT ADJ-WITH-IG
  ADJ
  (IG (AN ADJ
          (BASE :IF-NEEDED (SELF)
                (BASE (WHERE SELF = IG))
          (SUFFIX (AN IG-SUFFIX)))))
#<object ADJ-WITH-IG>

(DEFOBJECT IG-SUFFIX
  MORPHEME
  (LEXICAL-REPRESENTATION "+IG"))
#<object IG-SUFFIX>
```

This DEFOBJECT form is read as follows. *Adj-with-ig* is a category designating the prototypical adjective which has a (possible) derived form with *+ig*, stored as the value of an aspect *ig*. If one wishes to use a type theory terminology, *ig* can be interpreted as a function applying to objects of type *adj-with-ig*, with a return value of type *adj*. The derived form, i.e. the return value of the *ig* function, is an adjective with a *base* and a *suffix*. The *base* of the derived form is the same as the *base* of the object of which the derived form is derived (*where self = ig*). The *suffix* of the derived form is always an *ig-suffix*, i.e. a morpheme with the lexical representation *+IG*.

Other, more specific objects may inherit from the prototype *adj-with-ig* to access its method to perform the derivation. Each individual adjective inheriting from the prototype will have its own derived form. The set of objects inheriting from the prototype can thus be viewed as the domain of the rule.

---

[9] From the purely morphological point of view, we do not make a principled distinction between derivation and inflection, calling both processes by the name of derivation instead.

Again, following De Smedt (1984), these morphological classes are placed in a hierarchy, where new classes are formed by means of specialization and combination. The leaves of this hierarchy are the base words, i.e. the lexicon. For example, the Dutch adjective *groen* ('green') is in the domain of the *ig*-derivation. It is represented as an object which inherits from the category *adj-with-ig*; in addition it has a phonologically specified base:

```
(DEFOBJECT  GROEN
   ADJ-WITH-IG
   (BASE (A MORPHEME
            (LEXICAL-REPRESENTATION "#Grun"))))
#<object  GROEN>
```

This object will be dynamically modeled upon *adj-with-ig* by means of structured inheritance, and so the filler of the *ig* aspect will inherit from that in the prototype *adj-with-ig*, so that *groen* will have its own derived form with *+ig*. Since it was specified in the prototype that the base of the derived form is same as the base of the form of which it was derived, we have sufficient knowledge to establish that the derived form for this particular adjective is *groen+ig* (or *#Grun+IG*, phonologically). This adjective is created only when the generic function *ig* is applied to the object *groen*, as shown below.

```
(LEXICAL-REPRESENTATION  (IG  'GROEN))
(" "  "#Grun"  "+IG")
```

4.3 The range of one rule as the domain of another rule

Structured inheritance is especially useful when we want to specify that an object in the range of a rule is in the domain of another rule. For example, from *groen+ig* a comparative *groen+ig+er* ('greenisher') can be derived. Other possible orders of recursive word formation may be ungrammatical and must be ruled out (for example **groen+er+ig*). Using the functional metaphor, it makes sense to represent the result of the *ig*-derivation as an object which is in the domain of the comparative rule. This is schematically represented in Figure 9.

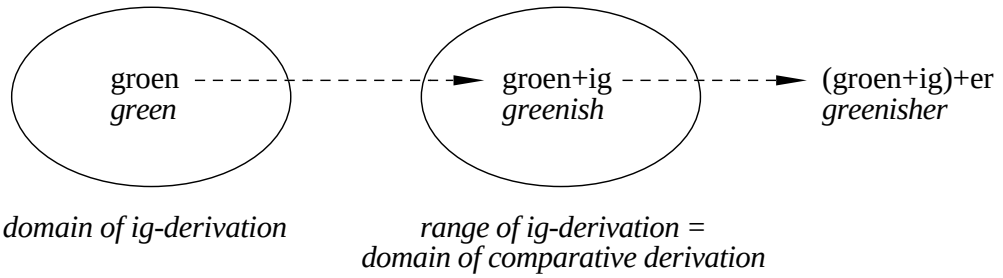

Figure 9

Words in the domains of rules

This knowledge can be represented in an object-oriented way as the constellation of objects in Figure 10, where gray dots represent objects automatically created by means of structured inheritance. Left to right order does not play a role in this figure.

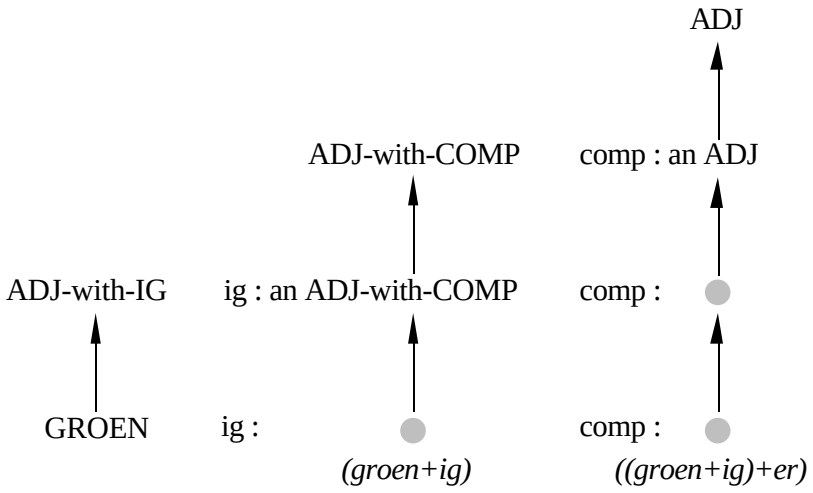

Figure 10

Structured inheritance creates derived forms and constrains them

The definition of *adj-with-ig* is thus adapted so that it constrains the result of the *ig*-derivation so that it is an adjective which may form a comparative:

```
(DEFOBJECT  ADJ-WITH-IG
  ADJ
  (IG (AN ADJ-WITH-COMP      ;may form comparative
          (BASE :IF-NEEDED (SELF)
                (BASE (WHERE SELF = IG))
          (SUFFIX (AN IG-SUFFIX)))))
#<object  ADJ-WITH-IG>
```

The form *groen+ig+er* is then created by a double derivation, i.e. an application of the generic function *comparative* to the result of the *ig*-derivation:

```
(IG 'GROEN)
#<an  ADJ-WITH-COMP>

(LEXICAL-REPRESENTATION (COMPARATIVE (IG 'GROEN))
(" " (" " "#Grun" "+IG") "+@r")
```

4.4 Valency reduction as a hierarchical phenomenon

The objects in the range of a derivation tend to have a smaller morphological valency than those in its domain, that is, their capacity to derive other word forms is more limited. This phenomenon is known as *valency reduction*. For example, *groen* can have both a derivation with *+ig*, i.e. *groen+ig*, and a comparative with *+er*, i.e. *groen+er*, while the *ig*-form itself cannot undergo another derivation with *+ig*, i.e. **groen+ig+ig*. Nor is it possible to reverse the order of the suffixes, i.e. **groen+er+ig*. Thus the base form *groen* has the highest valency, whereas its derived forms have ever diminishing valency. This instance of valency reduction is depicted in Figure 11.

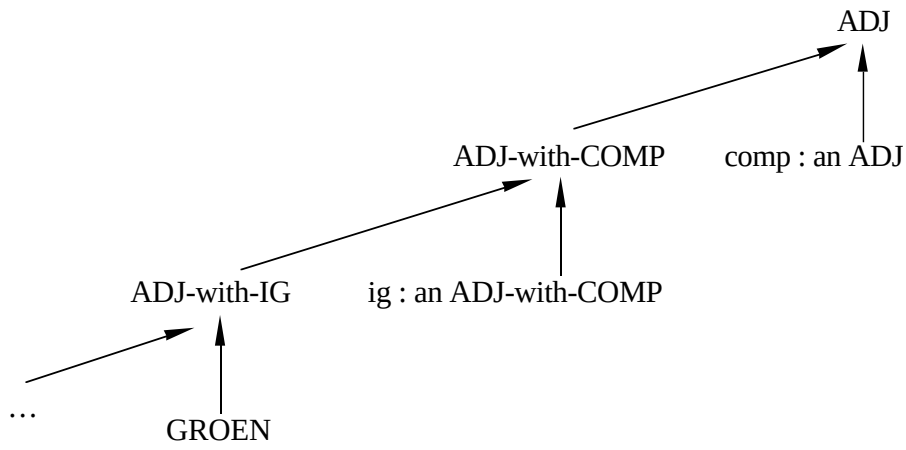

Figure 11

Valency reduction and the inheritance hierarchy

The highest object in the hierarchy has the lowest valency. Higher valency objects are formed by specializations which successively place the category in yet another rule domain. The base forms have the highest valency. Structured inheritance assigns a lower valency to their derived forms. At the same time, the organization of the hierarchy is one of the factors which account for the left-to-right order in which the morphemes occur in the derived forms.

Summing up, we have presented a minimally redundant way to organize a lexicon in a hierarchical way such that we account for the ordering constraints on suffixes. We have presented a classification of words in terms of the domains of the rules to which they belong. This rule-oriented organization is not the same kind of hierarchy as the morpheme-oriented organization which was presented in sections 2.3 and 3.3. A lexical object can conceivably be linked to both hierarchies.

4.5 Competing productive derivations and exceptions

Dutch has several possible suffixes for the formation of plural nouns. In addition to some non-productive paradigms, there are two competing productive paradigms, one with the suffix *+n* or *+en* and one with *+s*. Both have an open domain, where the domain of *+s* is marked by a number of conditions, and *+n/en* applies otherwise. The domains are therefore not represented by separate classes, but by a condition, here summarized as the predicate *conditions-for-s*: The CommonORBIT definition for a noun (N) which allows plural formation is the following:

```
(DEFOBJECT  N-WITH-PLURAL
  N
  (PLURAL (A N
              (BASE :IF-NEEDED (SELF)
                     (BASE (WHERE SELF = PLURAL));same as singular
              (SUFFIX :IF-NEEDED (SELF)
                       (COND ((CONDITIONS-FOR-S SELF)
                              (AN S-MORPHEME))
                             (T (AN EN-MORPHEME)))))))
```

Since structured inheritance is a form of default inheritance, exceptions specified in the lexicon will override the inherited information. For example, the Dutch noun *zee* ('sea') would normally get a plural with *+s*, but this must be overruled, because its plural is *zeeën*. This exception can be specified in a straightforward way, as shown in the following CommonORBIT code and depicted in Figure 12.

```
(DEFOBJECT ZEE
   N-WITH-PLURAL
   (BASE (A MORPHEME
            (LEXICAL-REPRESENTATION "#ze")))
   ((PLURAL SUFFIX) (AN EN-MORPHEME)))
#<object ZEE>

(DEFOBJECT EN-MORPHEME
   MORPHEME
   (LEXICAL-REPRESENTATION "+@n"))
#<object EN-MORPHEME>
```

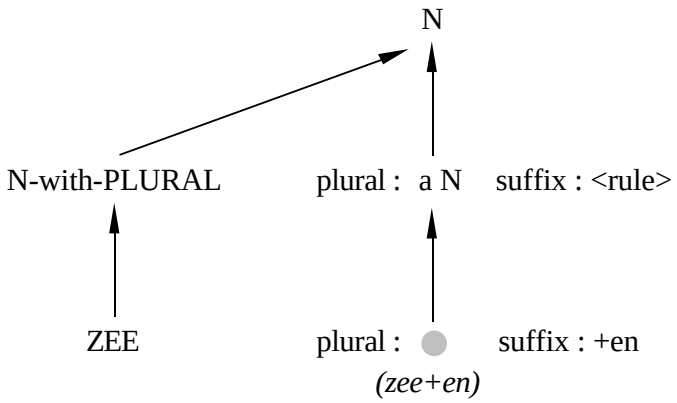

Figure 12

A default suffix predicted by structured inheritance is overruled

Similarly, the base of a plural noun form is generally equal to that of the singular, but there are some exceptions like *stad/sted+en* ('city/cities', with a kind of umlauting similar to German *Stadt/Städte*). We define the object *stad* so that the base of the plural is a morpheme with lexical representation *"sted"*. The exceptional nature of this lexical entry is depicted in Figure 13, where it can be seen that the default base of the plural noun is overridden.

```
(DEFOBJECT STAD
   N-WITH-PLURAL
   (BASE (A MORPHEME
            (LEXICAL-REPRESENTATION "#stAd")))
   ((PLURAL BASE) (A MORPHEME
                     (LEXICAL-REPRESENTATION "sted"))))
#<object STAD>
```

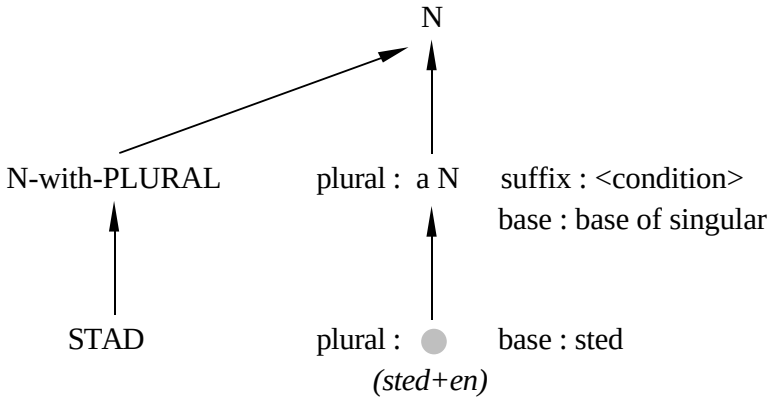

Figure 13

A default base predicted by structured inheritance is overruled

In a final and more intricate example, infixes are added by overriding a simple base with a structured one. Whereas the default suffix for female counterparts of person names is *+e*, the female counterparts of verbal derivations ending on *+er* are derived by infixation, for example *werk+st+er* ('female worker') from *werk+er* ('worker'), derived from the verb (V) *werk* ('work'). By giving the former the structure *((werk+st)+er)*, we reinterpret the infixation as a suffixing operation on the base. To this end, we define both the base and the suffix of the derived form as exceptions to the defaults, as shown in Figure 14.

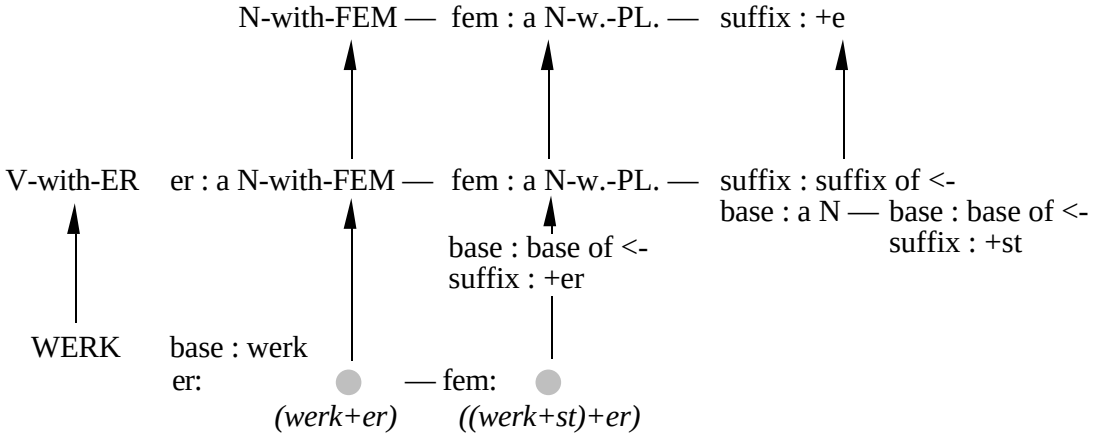

Figure 14

Infixing as the overruling of a default base with a structured base

In the schematic representation of Figure 14, the symbol '<-' denotes a pointer to the object to the left of the current one, i.e. the object in which the current object is an aspect filler. Using such paths of pointers, it can be seen that the *base* of the *fem* of the *er* of *werk* has the same *base* as *werk* itself.

5 FEATURE STRUCTURES AND INHERITANCE

We will now turn our attention to the representation of the grammatical knowledge needed for the construction of a syntactic representation of a sentence. Of the list of useful functions of inheritance which was presented in 1.2, *constraint checking* seems to be lacking. This is of course one of the main functions of *unification* in unification based formalisms (UBFs) (see Shieber, 1986 for an introduction to the literature on UBFs). However, it is possible to obtain the same functionality by letting multiple inheritance notify failure when contradicting information is inherited from different sources (multiple monotonic inheritance). This approach cannot be combined with default reasoning, however (unless when using two different inheritance mechanisms). Another approach to unification consist of the definition of a *unify* method that destructively merges objects to be unified.

5.1 Feature structures as objects

It is not difficult to see the analogy between UBFs and object-oriented formalisms. Feature structures can be represented as structured objects, and features as aspects (or slots). The following shows a CommonORBIT object definition and an equivalent bracketed representation of a feature structure for an instance of a noun phrase (NP):

```
(A  FEATURE-STRUCTURE
    (CATEGORY 'NP)
    (PLURAL '-)
    (NOMINATIVE '+))
```

$$\begin{bmatrix}\text{category} = & \text{NP} \\ \text{plural} = & - \\ \text{nominative} = & +\end{bmatrix}$$

Atomic feature values are objects without aspects, as well as other atoms such as symbols, strings and numbers. Recursive structures are represented by letting an aspect have a complex object as its value[10]. The following is an example:

```
(A  FEATURE-STRUCTURE
    (CATEGORY 'NP)
    (PLURAL '-)
    (NOMINATIVE '+)
    (HEAD
      (A ZEE))))))
```

$$\begin{bmatrix}\text{category} = \text{NP} \\ \text{plural} = - \\ \text{nominative} = + \\ \text{head} = \begin{bmatrix}\text{category} = \text{N} \\ \text{base} = [\text{lexical-rep} = \text{ze}\ ]\end{bmatrix}\end{bmatrix}$$

This example shows that objects can be referred to in the place of a full feature structure specification. In this case, an object which inherits from the noun *zee* (see Section 4.5) is created as a filler of the aspect *head.* Clearly, objects such as NP can be defined to function as

---

[10] Paths in recursive feature structures are a series of pointers from one object to another. Features in a path can be accessed by function application, for example:

```
(LEXICAL-REP (BASE (HEAD FS)))
```

Reentrancy (or feature sharing) is usually represented in UBFs by means of special labels; in object-oriented formalisms it is simply token identity.

templates for other ones. New objects that inherit from them can then easily be created. Using mixins as abbreviations makes the representation even more concise. For example, the definition of a mixin *singular* allows the feature structure above to be rewritten as follows; Figure 15 shows the part of the hierarchy which is involved here.

```
(A SINGULAR NOMINATIVE NP
   (HEAD (A ZEE)))
#<a client of SINGULAR, NOMINATIVE and NP>
```

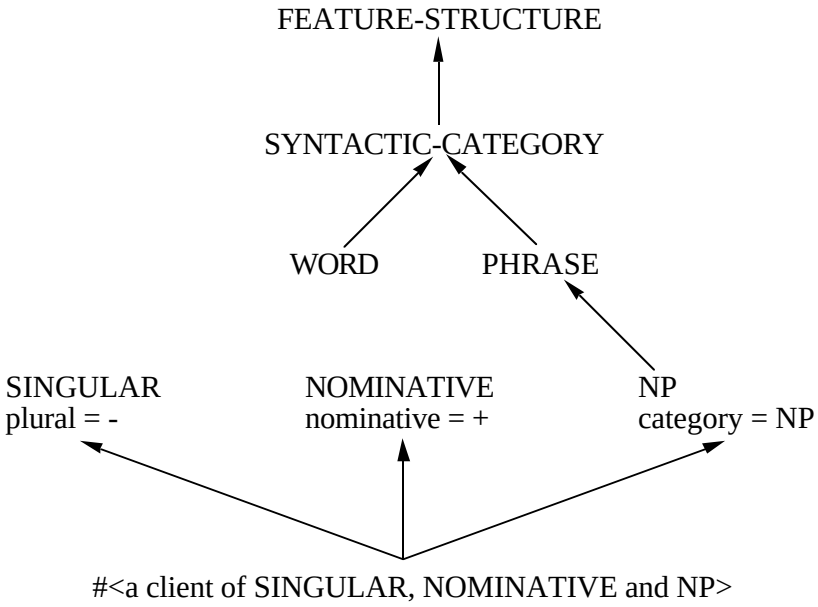

Figure 15

Example hierarchy of objects as feature structures

The necessity of being able to work with types, templates and any form of inheritance (if only to reduce redundancy) in UBFs has become increasingly clear. This has given rise to a number of monotonic or non-monotonic extensions or complementations of UBFs. Examples are *templates* and *overwriting* in PATR-II (Shieber, 1986:57-60), Kaplan's *priority union* (1987:180), Shieber's *add conservatively* (Shieber, 1986b), the use of datatyping and *sorts* in Unification Categorial Grammar (UCG, Moens et al., 1989), *sortal restrictions* in the Core Language Engine (CLE, Alshawi et al. 1989), feature structure and slot-filler *typing* in HPSG (Pollard and Sag, 1987), default reasoning in DATR (Evans and Gazdar, 1989), default unification (Bouma, 1992), and monotonic multiple inheritance in Typed Feature Structures (TFS, Zajac, 1992). Since default inheritance is already incorporated in most object-oriented languages, we take the reverse approach and represent features structures as objects, which allows them to use all the default reasoning machinery of existing object-oriented formalisms.

5.2 Segment Grammar

As a more concrete example of how inheritance can profitably be used in a UBF, we now briefly discuss an object-oriented representation of *Segment Grammar* (SG), a unification-

based formalism especially suited to incremental syntactic processing. Originally proposed by Kempen (1987), it has been further worked out and implemented by De Smedt & Kempen (1991) for the incremental generation system IPF. The basic units of SG are *syntactic segments* which represent individual syntactic relations between two categories. The top node of a segment in the context of a syntactic tree is called the *root* and the bottom node the *foot.* Typical examples are *NP-head-N*, representing the relation between an a noun phrase (the root) and its head noun (the foot), and *S-subject-NP*, representing the subject relation between an a sentence and a noun phrase.

Syntactic segments join together by means of a variant of the unification operation described above to form larger structures. The recursive[11] unification operation succeeds if the values of all these features in both objects match in one of the following ways: (1) If the values are both objects, then the objects are unified; the unification (if successful) becomes the new feature value; (2) otherwise, the disjunctive values are interpreted as sets and their intersection is computed; the intersection (if not empty) becomes the new feature value. If unification succeeds, then the two objects are merged into one, and the new feature values are stored into this one object, as well as all other information which was present in both objects.

SG has been implemented in CommonORBIT by uniformly representing all important grammar units—syntactic segments, syntactic categories (phrases, words) and syntactic features—as structured objects. Inheritance allows the grammar to be extended easily by creating segments as specializations or combinations of other ones. The specialization hierarchy of segments exploits multiple inheritance. For example, knowledge common to both segments S-subject-S and S-subject-NP is stored in a general segment S-subject-*. Likewise, knowledge common to all subordinate clauses is stored in *-*-S. By way of example, part of the segment hierarchy of a Dutch grammar is shown in Figure 16.

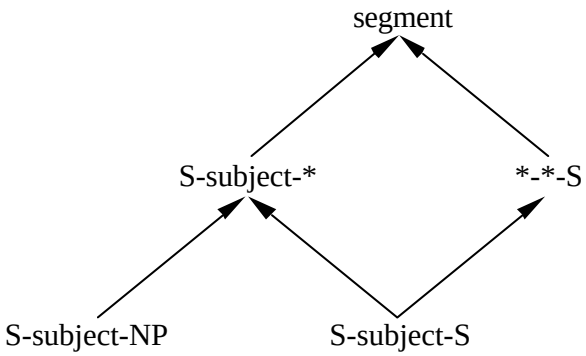

Figure 16

Part of the hierarchy of syntactic segments

[11] The full recursive power of unification in CommonORBIT is not strictly necessary in Segment Grammar, because grammatical relations, which require the recursion, are not represented in the same way as grammatical features, which are not recursive.

The structured inheritance mechanism in CommonORBIT establishes relations allowing inheritance between the root of a segment and that in its prototype, and likewise between the foot of a segment and that of its prototype. For example, consider the segment representing a NP with the head noun *water*. This segment—a lexical segment because it contains a word—inherits from NP-head-N. Consequently, relations for structured inheritance (see Section 4) are automatically established, as depicted in Figure 17. The noun *water* inherits from the N at the root of the NP-head-N segment. It thus obtains knowledge about the typical surface positions of a head noun (not explicitly shown in the example).

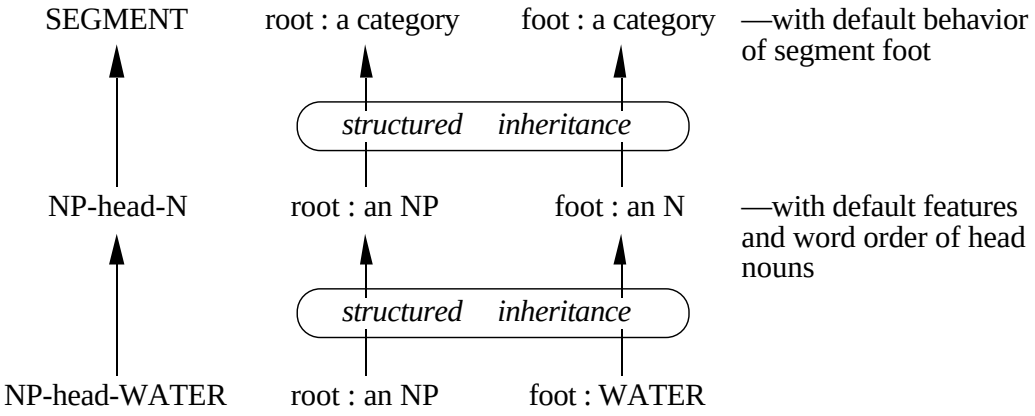

Figure 17

Structured inheritance in syntactic segments

Summing up, structured inheritance is a suitable mechanism for the representation of SG, which is a *lexicalized* grammar in the sense that the lexicon and the grammar are integrated. A full description of how this grammar is used in a sentence generation task falls outside the scope of this paper. For more details, the reader is referred to De Smedt and Kempen (1991).

6 MULTI-ATTRIBUTES

As a final illustration of the use of object-oriented languages in natural language processing, we discuss an extension of the object-oriented mechanisms that we have used so far. In unification-based approaches, knowledge is retrieved from feature structures by a process akin to function application. Applying the name of an attribute to a feature structure returns the value for that attribute in that feature structure[12]. In the previous section, we have pointed out the similarity of this to the object-oriented approach (instance variables are encapsulated within a

[12] We must add a note here about the functional nature of feature structures. Feature structures are usually seen as functions which map features onto values; in fact, they are sometimes called *functional* structures for this reason. In a sense, an object-oriented representation in CommonORBIT does the reverse in the sense that features are *generic* functions which map feature structures onto values. Some object-oriented languages (e.g. FLAVORS: Weinreb & Moon, 1980) do actually implement objects as functions.

single object). However, multiple default inheritance can be generalized in a way that is impossible to achieve in a unification-based approach. We introduce *multi-attributes*, inspired by multimethods in CLOS (Keene, 1989), as a means to associate attributes with *combinations* of objects. This provides us with an extremely powerful, minimally redundant, and notationally adequate way to describe linguistic generalizations in cases where many prototypes interact to determine a linguistic decision.

Consider as an example German weak adjectives. The choice of the suffix is determined by *case, number* and *gender*. The data in Table 1 are taken from Zwicky (1985). There are two possible suffixes: *+en* and *+e*. The suffix *+en* is the default. Direct (accusative or nominative) singular weak adjectives get *+e*, and an exception to this is constituted by the accusative masculine singular form, which gets *+en*. We have a situation here where the choice of the suffix cannot be predicted from any one of the morphological prototypes representing adjective classes, syntactic features, or suffixes, but only from the cooccurrence of a number of them.

Table 1

German weak adjective endings

		SING			PLUR		
		MASC	NEUT	FEM	MASC	NEUT	FEM
DIRECT	NOM	+e	+e	+e	+en	+en	+en
	ACC	+en	+e	+e	+en	+en	+en
OBLIQUE	GEN	+en	+en	+en	+en	+en	+en
	DAT	+en	+en	+en	+en	+en	+en

The only possible way to describe this with the type of inheritance discussed so far, would be either to use conditional statements in the method for the suffix, or to create *ad hoc* prototypes *direct-singular* as a subtype of *weak-adjective*, and *accusative-masculine* as a subtype of *direct-singular*. But flexibility increases considerably when multi-attributes can be associated with aggregates of prototypes. The following code (using the syntax of CLOS multi-methods) shows how the assignments would have to be formulated, and two tests for the weak adjective *breit.*

```
(DEFMETHOD SUFFIX ((ADJ WEAK) CASE NUMBER GENDER)
   '+EN)

(DEFMETHOD SUFFIX ((ADJ WEAK)(CASE DIRECT)(NUMBER SING) GENDER)
   '+E)

(DEFMETHOD SUFFIX ((ADJ WEAK)(CASE ACC) NUMBER (GENDER MASC))
   '+EN)

(SUFFIX BREIT NOM SING MASC)
+E
```

```
(SUFFIX BREIT GEN PLUR MASC)
+EN
```

With three assignments, we have described all the relevant data (involving 24 possible combinations of feature values for each weak adjective) without the creation of any spurious prototypes. The power of multi-attributes derives from the flexibility they provide in accessing arbitrary regions in a multi-dimensional space formed by different feature hierarchies, while at the same time allowing default reasoning. They also allow the use of several independent hierarchies where otherwise one deep and tangled hierarchy would have to be used.

Another example of the expressive power of multi-attributes is the description of German separable verbs. In Russell et al. (1992), a default inheritance treatment of this phenomenon is presented. We will adopt most of the prototypes and general organization used there, and restrict ourselves to demonstrating how multi-attributes can be used to improve notational adequacy.

German separable verbs are a subtype of prefixed compound verbs in which the prefix is a bound morpheme both when the verb is untensed and when it is the head of a verb-final clause. This situation is marked by a feature *INV = no* . In the solution of Russell et al., this is described by associating three *variant sets* of feature equations with the class separable corresponding with the situations where the verb is *untensed*, *tensed* and *INV = no*, and *tensed* and *INV = yes.* In the latter case, the prefix of the verb is empty (i.e. a null morpheme). In Figure 18, a hierarchy is shown which is similar to the one in Russell et al., but with two small independent hierarchies for tensedness and inversion.

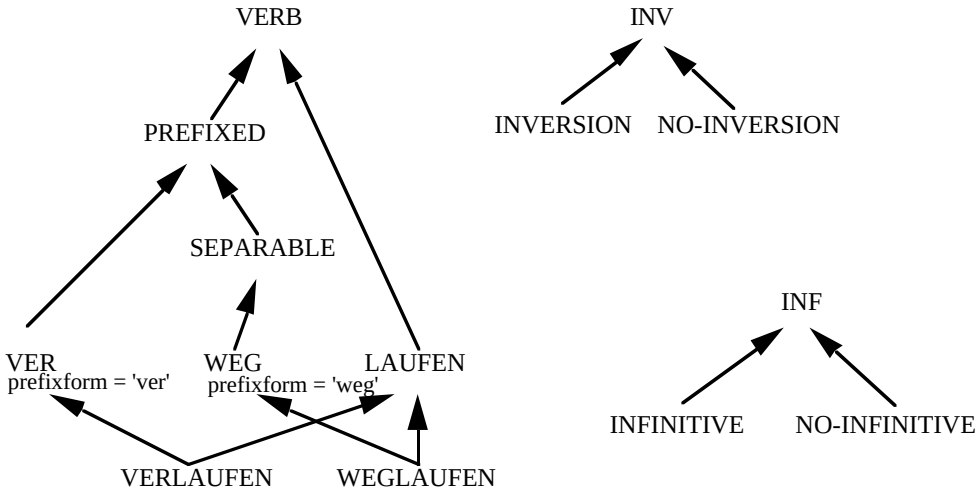

Figure 18

Hierarchies for German separable verbs

The influence of syntactic context on the separateness of the prefix from the verb can now be easily described using the following rules, again using the syntax of CLOS.

```
(DEFMETHOD PREFIX (VERB INV FIN)
  (A 0-MORPHEME))
```

```
(DEFMETHOD PREFIX ((VERB PREFIXED-VERB) INV FIN)
  (PREFIXFORM SELF))

(DEFMETHOD PREFIX ((VERB SEPARABLE-VERB) (INV INVERSION)
                   (FIN INFINITIVE))
  (A 0-MORPHEME))
```

The first multi-attribute definition states that all verbs whatever their syntactic context lack a prefix. The second default states that prefixed verbs have as prefix the prefix-form of their prefix, again whatever the syntactic context. Finally, the third multi-attribute states that separable prefixed verbs when they are tensed and at the same time occur as head of a non-verb-final clause have no prefix (i.e. the prefix is a null morpheme).

7 CONCLUDING REMARKS

We have shown that natural language is a knowledge domain which benefits from the use of object-oriented techniques. We have demonstrated how different forms of inheritance can be applied for different purposes. We have treated the use of inheritance not only for specialization, but also for combination of defaults by means of multiple inheritance, and we have shown how structured inheritance can predict the behavior of new objects which are created by other objects. The resulting elimination of redundancy should not merely be seen as a way to save memory resources, but more importantly as the basis for abstraction in linguistic theory. Our examples have shown how traditional linguistic notions, such as exceptions and blocking, can be transparently modeled by means of hierarchical reasoning with defaults. All of this can be programmed in existing object-oriented languages.

Several similar approaches to achieve generalizations can be found in the literature. For a general overview of the use of inheritance in natural language processing, we refer the reader to Daelemans, De Smedt and Gazdar (1992). Most of this research, however, is devoted to formal aspects of inheritance and to special-purpose inheritance formalisms. In contrast, we have given several illustrations of representations in existing object-oriented languages which are not specially meant for representing linguistic knowledge, but which nevertheless provided suitable and powerful mechanisms. This illustrations have served to substantiate our point that the creation of special linguistic formalisms with similar capacities is unneccessary and wasteful. Furthermore, it is almost implicit in the use of a domain-independent representation that linguistic and non-linguistic knowledge share a common basis for their representation. Indeed, this uniformity of representation makes it possible to bring out and exploit commonalities in different cognitive domains. We see no reason why the operation of default inheritance in a linguistic domain would be different from that in other cognitive domains.

Nevertheless, some general problems with an object-oriented approach remain. One problem—in fact shared by all symbolic approaches to natural language processing—is that there are many alternative ways in which a domain can be modeled. The designer has to make explicit choices about which object types, attributes, and taxonomies to choose. Furthermore, once designed, models are fairly inflexible and rigid because they make domain commitments

that are hard to change. One of the main concerns for future research in inheritance-based natural language processing (as well as all other inheritance-based reasoning) should therefore be the development of techniques for automatically acquiring and adapting specialization hierarchies.

A second area for future research is the extension of object-oriented formalisms with notions from other network-based paradigms, such as connectionist models. For example, object-oriented formalisms could be extended by associating activation levels with objects. These activation levels could be used to dynamically determine the precedence of prototypes for multiple inheritance. This is useful to model contextual influences on word sense disambiguation, and misclassification (for example overgeneralization) in morphology. Attributes could be given an activation level as well, which makes a dynamic definition of object equality possible. Activation of a particular object or attribute may be the result of contextual bias, it may be relative to frequency, or to any other notion of salience. Incorporating the notion of activation into an object-oriented model supports hybrid symbolic-associative models, combining the strengths of both approaches.

REFERENCES

Alshawi, H., Carter, D., Van Eijck, J., Moore, R., Moran, D., Pereira, F., Pulman, S. & Smith, A. (1989). *Final report: Core language engine* (Technical report, Project no. 2989). Cambridge, MA: SRI.

Bobrow, R.J. & Webber, B.L. (1980). Knowledge Representation for Syntactic/Semantic Processing. In *Proceedings of the First Annual National Conference on Artificial Intelligence,* 316-323. Stanford, CA: Stanford University.

Bouma, G. (1992). Feature structures and nonmonotonicity. *Computational Linguistics, 18,* 183-203.

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

Brewka, G. (1989). Nonmonotonic logics—a brief overview. *AI Communications 2,* 88-97.

Daelemans, W. (1987). *Studies in language technology: An object-oriented model of morphophonological aspects of Dutch.* Ph.D. dissertation, University of Leuven, Department of Linguistics.

Daelemans, W. (1988). A model of Dutch morphophonology and its applications. *AI Communications, 1(2),* 18-25.

Daelemans, W. (1990). Inheritance in Object-Oriented Natural Language Processing. In W. Daelemans & G. Gazdar (Eds.), *Proceedings of the First Workshop on Inheritance in Natural Language Processing, Tilburg, 16-18 August 1990* (pp. 30-38). Tilburg: University of Tilburg, Institute for Language Technology and Artificial Intelligence.

Daelemans, W., De Smedt, K. & Gazdar, G. (1992). Inheritance in natural language processing. *Computational Linguistics, 18,* 205-218.

De Smedt, K. (1984). Using object-oriented knowledge-representation techniques in morphology and syntax programming. In T. O'Shea (Ed.), *Proceedings of the 6th European Conference on Artificial Intelligence* (pp. 181-184). Amsterdam: Elsevier.

De Smedt, K. (1987). Object-oriented programming in Flavors and CommonORBIT. In R. Hawley (Ed.), *Artificial Intelligence programming environments* (pp. 157-176). Chichester: Ellis Horwood.

De Smedt, K. (1989). *Object-oriented knowledge representation in CommonORBIT* (Internal Report 89-NICI-01). Nijmegen: University of Nijmegen, Nijmegen Institute for Cognition research and Information technology (NICI).

De Smedt, K. & De Graaf, J. (1990). Structured inheritance in frame-based representation of linguistic categories. In W. Daelemans & G. Gazdar (Eds.), *Proceedings of the First Workshop on Inheritance in Natural Language Processing, Tilburg, 16-18 August 1990* (pp. 39-47). Tilburg: University of Tilburg, Institute for Language Technology and Artificial Intelligence.

De Smedt, K. & Kempen, G. (1991). Segment Grammar: A formalism for incremental sentence generation. In C. L. Paris, W. R. Swartout & W. C. Mann (Eds.), *Natural language generation in artificial intelligence and computational linguistics* (pp. 329-349). Dordrecht: Kluwer Academic Publishers.

Ducournau, R. & Habib, M. (1987). On some algorithms for multiple inheritance in object-oriented programming. In *Proceedings of ECOOP'78* (Bigre+Globule 54) (pp. 291-300). Paris: AFCET.

Evans, R. & G. Gazdar. (1989). The semantics of DATR. In *Proceedings of the EACL, Manchester.* Morristown, NJ: ACL.

Fikes, R. & Kehler, T. (1985). The role of frame-based representation in reasoning. *Communications of the ACM , 28,* 904-920.

Flickinger, D. (1987). *Lexical Rules in the Hierarchical Lexicon.* Ph.D. dissertation. Stanford University, Department of Linguistics.

Frazer, N.M. & Hudson, R.A. (1992). Inheritance in Word Grammar. *Computational Linguistics, 18,* 133-158.

Gazdar, G. (1987). Linguistic applications of default inheritance mechanisms. In P. Whitelock, M. Wood, H. Somers, R. Johnson & P. Bennett (Eds.), *Linguistic Theory and Computer Applications* (pp. 37-67). London: Academic Press.

Hudson, R. (1984). *Word Grammar.* Oxford: Basil Blackwell.

Jacobs, P.S. (1985). *A knowledge-based approach to language production.* Ph..D. dissertation, University of California, Berkeley.

Kaplan, R. (1987) Three seductions of computational psycholinguistics. In P. Whitelock, M. Wood, H. Somers, R. Johnson & P. Bennett (Eds.), *Linguistic Theory and Computer Applications* (pp. 149-188). London: Academic Press.

Keene, S. (1989). *Object-oriented programming in Common Lisp* Reading, MA: Addison-Wesley.

Kempen, G. (1987). A framework for incremental syntactic tree formation. *Proceedings of the 10th IJCAI* (pp, 655-660). Los Altos: Morgan Kaufmann.

Lieberman, H. (1986). Using prototypical objects to represent shared behavior in object-oriented systems. (Proceedings of the First ACM Conference on Object-Oriented Programming Systems, Languages, and Applications). *SigPlan Notices, 21,* 214-223.

Moens, M., Calder, J., Klein, E., Reape, M. & Zeevat, H. (1989). Expressing generalizations in unification-based grammar formalisms. *Proceedings of the 4th European ACL Conference, (pp. 174-181).* Morristown, NJ: ACL.

Pereira, F. & Shieber, S. (1984). The semantics of grammar formalisms seen as computer languages. *Proceedings of the 10th Coling* (pp. 123-129). Morristown, NJ: ACL.

Pollard, C. & Sag, I. (1987). *Information-based syntax and semantics. Volume 1: Fundamentals* (CSLI Lecture Notes 13). Stanford, CA: CSLI.

Russell, G., Ballim, A., Carroll, J. & Warwick-Armstrong, S. (1992). A practical approach to multiple default inheritance for unification-based lexicons. *Computational Linguistics, 18,* 311-337.

Steels, L. (1978). *Frame-based knowledge representation* (Working Paper 170). Cambridge, MA: MIT AI Laboratory.

Shieber, S.M. (1986). *An introduction to unification-based approaches to grammar* (CSLI Lecture Notes 4). Chicago: University of Chicago Press.

Shieber, S. M. (1986b). A simple reconstruction of GPSG. *Proceedings of COLING 1986* (pp. 211-215). Bonn, Germany.

Touretzky, D.S., Horty, J.F. & Thomason, R.H. (1987). A clash of intuitions: The current state of nonmonotonic multiple inheritance systems. *Proceedings of the 10th IJCAI* (pp. 476-482). Los Altos: Morgan Kaufmann.

Van der Linden, E., Brinkkemper, S., De Smedt, K., Van Boven, P. & Van der Linden, M. (1989). The representation of lexical objects. In T. Magay & J. Zigány (Eds.), *Proceedings of the EURALEX Third International Congress.* Budapest: Akadémiai Kiaido. (Also published as Internal Report 88-ITI-B-33. Delft: TNO).

Van Marcke, K. (1987). KRS: An object-oriented representation language. *Revue d'Intelligence Artificielle, 1*(4).

Weinreb, D. and Moon, D. (1980). *Flavors: message passing in the Lisp Machine (*Memo AIM-602). Cambridge, MA: MIT.

Wirth, N. (1971). Program development by stepwise refinement. *Communications of the ACM, 14,* 221-227.

Zajac, R. (1992). Inheritance and constraint-based grammar formalisms. *Computational Linguistics, 18,* 159-182.

Zwicky, A. (1985). How to describe inflection. In *Proceedings of the Berkeley Linguistic Society.*