Natural Language Interface Framework for Spatial Object Composition Systems*

Hiram Calvo Castro [1] and **Alexander Gelbukh** [1,2]
[1] Center for Computing Research (CIC), National Polytechnic Institute (IPN),
Av. Juan de Dios Bátiz s/n, esq. Av. Mendizábal, México, D.F., 07738. México
hcalvo@sagitario.cic.ipn.mx, gelbukh@cic.ipn.mx; www.gelbukh.com
[2] Chung-Ang University, Seoul, Korea

Resumen: Los Sistemas de Composición Espacial de Objetos (Sistemas SOC, por sus siglas en inglés) involucran tareas de combinación virtual de objetos físicos (como partes de muebles) con el propósito de crear objetos complejos nuevos o disponer los objetos en el espacio. En este artículo presentamos un marco para implementar Interfaces en Lenguaje Natural enfocadas a sistemas SOC. Proponemos el uso de una gramática de reglas de reescritura, a la cual llamamos Gramática de Traducción Directa, para traducir solicitudes en lenguaje natural a comandos computacionales interpretables por el motor del Sistema SOC. En este artículo damos ejemplos para comandos imperativos en español.
Palabras clave: Lenguaje Espacial, Solicitudes de Acción, Interacción con Diálogos.

Abstract: Spatial Object Composition (SOC) systems involve tasks of virtual combination of physical objects (such as furniture parts) with the purpose of creating new, complex objects or arranging the objects in space. In this paper we present a framework for implementing Natural Language Interfaces focused on SOC systems. We propose the use of a rewriting rules grammar (which we call Direct Translation Grammar) to translate action queries in natural language to computational procedures interpretable by the SOC engine. Examples are given for imperative commands in Spanish.
Keywords: Spatial Language, Action Queries, Dialogue Interaction

1 Introduction

Spatial Object Composition (SOC) refers to manipulating physical or virtual prefabricated pieces (such as furniture parts) to assemble them creating new objects or scenes, for example, is office design task. There are many computer applications dealing with SOC, for example, the systems for computer-aided design of a room or a house. The objects that are to be placed in the room are predefined (furniture, doors, windows, etc.) and can be selected from a catalog to be placed in the virtual scene where the user wants to place them.

Obviously, SOC is not limited to house design. Since we live in a spatial world of decomposable objects, there are many applications of this kind. For example, suppose a user wants to construct a bookcase. To do this, he or she first selects some planks from a catalog of prefabricated planks and then fits these parts together until the desired bookcase is constructed. This is an example of creating a new object.

By their nature, such systems are intended to be used by persons without any computer-related knowledge and skills. Thus their interfaces must be intuitive and must not require any training or instruction. A perfect means of such interaction is natural language in the same form as it would be used for interaction with a human worker. Indeed, human-computer interaction in such systems is mostly imperative: the user gives a command and the computer executes the requested task. These commands can be given in natural language using imperative tense. Hence our motivation to develop a

* Work done under partial support of Mexican Government (CONACyT, SNI), IPN (PIFI, CGEPI), and RITOS-2. The second author is currently on Sabbatical leave at Chung-Ang University.

framework for integration of Natural Language Interfaces with SOC systems.

Within the proposed framework it is possible to translate the input sentence "*Could you put the chair next to the table, please?*" into a sequence of commands directly interpretable by the system's engine:

```
move(obj_chair1,getpos(obj_table1)++)
```

Here, `obj_x` stands for an object *x*, `getpos` for a function that gets the position of the object, and `++` for the operation that changes a position to the nearest available one. The whole instruction means that the system must place `obj_chair1` (the chair the user referred to) at the nearest position available next to that of `obj_table1` (the table already existing in the scene). We do this by transforming the original sentence step by step as follows:

Could you put the chair next to the table, **please?**
Could you put the chair next to the table
put **the chair** next to the table
put obj_chair1 next to **the table**
put obj_chair1 **next to** obj_table1
put obj_chair1 **nextto obj_table1**
put obj_chair1 **(getpos(obj_table1)++)**
move(obj_chair1,(getpos(obj_table1)++))

In this paper, we describe the formalism we developed for such transformation, which includes the features that, in our experience, are necessary for successful translation of such type of sentences. Unfortunately space limitations will not allow us to give a meaningful example of their application.

The paper is organized as follows. Section 2 discusses the related work on Natural Language Interfaces (NLIs). Section 3 examines the characteristics of SOC systems relevant for their integration with a NLI. Section 4 introduces the grammar used in our framework, called Direct Translation Grammar. Section 5 explains the technicalities of object reference and context management in our grammar. In Section 6, a simple example is given. Finally, in Section 7 conclusions are drawn.

2 Related Work

Historically first systems with a Natural Language Interface (NLI) were developed on the *ad hoc* basis for a specific application. Some examples of such systems are:

- DEACON (Direct English Access Control) (Craig *et al.*, 1966), a question answering system,
- SHRDLU (Winograd, 1972), allowing to move virtual geometric blocks by verbal commands,
- LUNAR (Woods, Kaplan and Nash-Webber, 1972), which allowed to query a lunar rock database,
- LADDER (Hendrix *et al.*, 1978), which answers questions about naval logistics data.

As the world model was interwoven in these programs' operation, changing the application domain for these systems would be an expensive and complicated process.

Later, other systems with a NLI designed with a broader scope of application arose. They were mainly oriented to database information retrieval, e.g.: INTELLECT (Harris, 1984), TEAM (Grosz *et al.*, 1987), JANUS (Weischedel, 1989), and SQUIRREL (Barros, 1995).

There are recently developed works that handle imperative language for multiple purposes. For example, KAIRAI (which means 'puppet') has several virtual robots (avatars) that can move forward, turn, or push an object (Shinyama, Tokunaga and Tanaka, 2000; Asoh *et al.*, 1999). By manipulating them using commands, the user can move and place the objects in the virtual world. This system is developed for Japanese. A similar system AnimAL uses a NLI to control the movements of an avatar in a virtual environment (Di Eugenio, 1993; 1996; Webber, 1995). Di Eugenio considered the problem of understanding phrases of the form *do x to do y*, as in *cut a square in half to make two triangles*.

We are not aware, however, of any recent works specifically devoted to provide a NLI framework for SOC systems in general.

3 Characteristics of SOC Systems

The SOC systems in general restrict the use of natural language in a number of ways. In our framework, we rely on these restrictions to simplify the corresponding mechanisms. Specifically, SOC systems have the following characteristics relevant for NLI design:

1. **They have predefined basic objects that can be used to construct new ones.** This permits us to begin with a reduced set of object names to be recognized.

2. **Objects have properties** by which they can be referred to, e.g., *red plank* as opposed to *green plank*. Properties let us keep small our set of object names.
3. **There is a visual spatial representation common to the user and the computer.** With this, the user is aware that the only existing objects are those that can be observed in the catalogues and in the current scene. Only the observable objects are relevant for the composition task.
4. **Objects have a limited number of actions that can be applied to them**. They can be mapped to the corresponding computer commands.

The user and the computer manipulate a finite set of objects with properties and actions attached to these objects. To design a suitable NLI, we must find a mechanism that relates natural language sentences with the corresponding computer commands. This relation is implemented through Direct Translation Grammar presented in the next section.

4 Direct Translation Grammar

Since the transformational model by Chomsky appeared in 1957 (Chomsky, 1957), a number of models within the generative paradigm have been suggested, such as Case Grammar (Fillmore, 1968), Functional Grammars (Kay, 1979), and recently, Phrase Structure Grammars (Gazdar, 1987; Sag and Wasow, 1999). Traditionally, generative grammars are designed to model the whole set of sentences that a native speaker of a natural language considers acceptable (Pullum, 1999). Generative linguistics views language as a mathematical object and builds theories similar to the sets of axioms and inference rules in mathematics. A sentence is grammatical if there is some derivation that demonstrates that its structure corresponds to the given set of rules, much as a proof demonstrates the correctness of a mathematical proposition (Winograd, 1983).

Phrase Structure Grammars (PSG), from which HPSG (Sag, 1999) is the most widely known, follow this generative paradigm. To analyze a sentence, it is hierarchically structured to form phrase-structure trees. PSGs are used to characterize these phrase-structure trees. These grammars consist of a set of non-terminal symbols (phrase-structure categories such as Noun, Verb, Determiner, Preposition, Noun Phrase, Verbal Phrase, Sentence, etc.), a set of terminal symbols (lexical items such as *buy*, *John*, *eaten*, *in*, *the*, etc.), and a set of rules that relate a non-terminal with a string of terminal or non-terminal symbols (Joshi, 1992). To analyze a sentence, suitable rules can be applied to the terminal symbol string until the non-terminal symbol S is reached. The phrase-structure tree obtained during this process can be analyzed later to generate computer commands equivalent to the input sentence.

However, this process can be done directly if we change the purpose of our grammar to that of using the grammar rules to reach computer commands directly instead of breaking natural language sentences into parts of speech (phrase structures) and then converting this structure to computer commands. Thus our purpose is different from that of generative grammars in that we are not interested in determining whether or not a sentence is well-formed. In addition, we are not interested in modeling the whole language but only its small subset relevant for the user's task in question.

The grammar we suggest to translate natural language sentence into computer commands is a rewriting rules grammar with additional characteristics to handle context and object reference. We call this grammar Direct Translation Grammar (DTG).

Within DTG, lexical and morphological treatment is included, and the categories used refer to syntactic and semantic concepts of the sentences. Because of this we can consider DTG a semantic grammar (Burton, 1992). In semantic grammars, the choice of categories is based on the semantics of the world and the intended application domain, as well as on the regularities of the language. Although they are not widely used nowadays, semantic grammars have several advantages such as efficiency, habitability—in the sense of (Watt, 1968), handling of discourse phenomena, and the fact that they are self-explanatory. They allow using semantic restrictions to reduce the number of alternative interpretations that can be considered at a certain moment, in contrast to highly modular systems, which fragment the interpretation process.

4.1 Definition

We define a Direct Translation Grammar as an ordered list of rewriting rules that have the form $\alpha \rightarrow \beta$, where α and β are strings consisting of

Notation	Property	Possible values
C	category	N (noun), V (verb), ADJ (adjective), ADV (adverb), PRO (pronoun), DEFART (definite article), INDART (indefinite article), OBJ (object), POS (position)
G	gender	M (masculine), F (feminine), N (neutral)
N	number	S (singular), P (plural)
T	verbal tense	PRES (present), INF (infinitive), IMP (imperative), SUBJ (subjunctive)
S	subject form	for verbs, the number and gender of the subject (this is morphologically relevant for Spanish): SM, SF, PM, PF (singular / plural, masculine / feminine)
O	object form	for verbs: the number and gender of the object (morphologically relevant for Spanish)
A	dative object form	for verbs: the number and gender of the indirect (dative) object (morphologically relevant for Spanish)
Q	quantity	V, L, R, U, M (very little, little, regular, much/many, very much/many)

Table 1. Some properties and their values used in the examples

one or more of the following elements (which we explain below) in any order:

1. natural language words,
2. tags with properties,
3. wildcards,
4. names of external procedures,
5. symbolic references to objects, and
6. embedded functions for context control and object reference handling, see Section 5.

Two rules with the same α are not allowed.

4.2 Rule Order

Since several rules can be applicable to a string at the same time, processing of the rules is ordered. First, the rules with α consisting only of natural language words are considered, beginning with those with a greater number of words. If none of them can be applied, the rest of the rules are considered according to the number of elements that form α, longest ones being considered first. This is because the elements like *the red table* must be considered before the elements containing just *the table*. Indeed, a longer string of words means a more specific reference to an object.

Each time a rule is applied, the processing of the rules restarts from the top of the list in the order just explained.

The process finishes when no rule can be applied; the resulting string is the output of the program. The translation process is considered successful if the resulting string consists only of symbolic references to objects and names of external procedures. To avoid infinite cycling, the process is aborted if some rule is applied more than once and its application results in a previously obtained string; in this case translation is considered unsuccessful, and the user is asked to rephrase his or her utterance.

4.3 Rule Components

In this section we explain each element used in the rules, in the order in which they are listed in Section 4.1.

4.3.1 Natural language words

Initially, an input sentence consists only of words. The example *put the chair next to the table* is a sentence composed by 7 words that will be translated into a sequence of computer commands. Words are letter strings and do not have any properties.

4.3.2 Tags with properties

Tags with properties have the form

$$\delta\{p_1, p_2, \ldots, p_n\},$$

where δ is the name of the tag and p_1, p_2, ..., p_n its properties in the form `name:value`, e.g. `put{C:V, T:IMP}`. Table 1 presents the most common properties and their possible values.

This construction resembles the traditional feature structures. However, feature structures, as defined by Kay (1979), undergo inheritance mechanisms and unification. Our tags are not related to such mechanisms.

For example, the following rule converts the Spanish word *pon* 'put please' into a tag `poner` 'to put':

```
pon --> poner{C:V, T:IMP, S:2S, A:1S}
```

'$\text{put}_{\text{imperative}}$ --> to put'

This rule substitutes every occurrence of *pon* in the input string by the tag `poner{C:V, T:IMP, S:2S, O:1S}`, whose properties are interpreted as follows: category is verb, tense is imperative, subject is of second person singular, (implicit) dative object is of first person singular.

4.3.3 Wildcards

Wildcards are defined by a label optionally followed by a set of properties (as defined in Section 4.3.2) contained in square brackets:

$\varphi[p_1, p_2, ..., p_n]$.

They provide a mechanism for generalizing a rule to avoid redundant rule repetitions. A wildcard makes it possible to apply a rule over a set of tags that share one or more properties. The scope of a wildcard is limited to its rule.

A wildcard φ matches with a tag δ if the δ has all properties listed for φ and with the same values. For example, both wildcards `A[C:V]` and `B[T:IMP, S2S]` match with the tag `poner{C:V, T:IMP, S:2S, O:1S}`, but `C[C:V, T:PRES]` does not, since this tag does not have the property *Tense* with value *Present*.

When used in the right-hand side of the rule, a wildcard can be used to modify properties by specifying another value for the property that it originally matched. For example, consider the frequently used pair of words *podrías juntarlo* 'could you please put it together', which is a polite euphemism for the imperative *júntalo* '$\text{put}_{\text{imperative}}$ it together'. To transform it into imperative, we first apply the following rules:

```
podrías --> poder{C:V, T:SUBJ, S:2S}        (1)
'could you --> can'
juntarlo --> juntar{C:V, T:INF, O:3SM}      (2)
'put it together --> to put together'
```

and then use a wildcard to transform any such construction into an imperative; note the use of a wildcard to change the property T from INF to IMP:

```
poder{C:V,T:SUBJ,S:2S} A[C:V, T:INF]
'can'                          --> A[T:IMP]  (3)
```

which results in the following output string:

```
juntar{C:V,T:IMP,O:3SM}                     (4)
'to put together'
```

Due to the wildcards, the rule (3) works for any polite expression in the form *podrías* 'could you' + infinitive verb.

Usually, properties found within brackets are accessed for the object whose name appears immediately to the left of these brackets. However, access to the properties for other objects outside brackets is possible through the use of the dot notation defined as follows. Consider the following string:

```
juntar{C:V,T:IMP,O:3SM} un poco más
'put it together'          'a bit more'
```

the collocation *un poco más* 'a bit more' is transformed into a quantity adverb by the rule

```
un poco más --> x{C:ADV, Q:L},              (5)
```

which then is transformed into the verb's property by the rule:

```
A[C:V] B[C:ADV,Q] --> A[Q:B.Q].             (6)
```

This rule sequence means the following: if a verb A is followed by an adverb B with some quantity, then add to this verb the property Quantity with the same value that it has in B. The latter construction is expressed in (6) as `B.Q` standing for the value of `Q` in `B`.

If a property is specified for a wildcard without any value, this indicates that matching the wildcard requires the property to be present regardless of its value.

Note that due to this replacing capability wildcards are not reduced to unification of properties (Knight, 1992).

4.3.4 External procedures

External procedures with arguments are formed by a procedure name followed by arguments:

`procedure_name (`arg_1`, `arg_2`, ... , `arg_n`)`,

where *n* is a natural number. This number can be 0; in this case the procedure has no arguments. Unlike functions, procedures do not return any value. They are executed by the SOC system's engine after successful application of rules over an utterance. For example, `move(A,B)` is an external procedure that places object `A` in position `B`.

4.3.5 Symbolic references to objects

A scene is an object composed by other objects. In their turn, these objects can be composed of other objects. For example, catalogs are objects composed of elements that are objects as well.

Such compositionality permits us to establish nested contexts to resolve the reference to an object depending on the scene in the focus of

Function	Description
`s.push (x)`	pushes the object *x* onto the stack *s* and returns an empty string
`s.pop ()`	pops the top object from the stack *s* and returns an empty string. If the stack was empty, returns NIL
`object s.top ()`	returns the object name from the top of the stack *s* without popping it. If the stack was empty, returns NIL
`object s.search (s, p = v)`	searches for the first object with the value *v* of the property *p*, starting from the top of the stack *s*. If no object is found, returns `NIL`.

Table 2. Embedded functions and procedures.

the user's attention in a given moment. Each one of the objects inside the scene has properties that can be accessed by our conversion rules by means of the tags.

In contrast to grammatical properties, which are described exclusively within our conversion rules, object properties are defined by the SOC system and can vary. These properties can be, for example, position, size, components, color, material, alterability, shape, and a set of actions that can be applied to the given object.

Labels beginning with `obj_` denote symbolic references to objects, e.g., `obj_box123` refers to a particular box appearing in a particular scene.

5 *Object Reference and Context Management*

For each noun, pronoun, or noun phrase we need to find a unique symbolic reference to a particular object meant by the user. However, the same expression (as string of letters) can be used to refer to different particular objects, depending on the context. To transform an expression into a symbolic reference, we should first determine the context for it (Pineda, 2000).

To provide context handling, we consider context as an object (called scene object) that contains other objects. A context change occurs when the user shifts his or her attention from the object itself to its components, or vice versa. E.g., the user can consider a catalog, or objects from this catalog, or parts of specific objects from the catalog. Here we can see that catalog objects belong to one context (the catalog), while objects in it belong to another context. Each of these contexts is called a scene.

Similarly to SQUIRREL (Barros, 1995), in our model context and object reference are managed by stacks. However, in contrast to SQUIRREL, we allow the grammar to create and manipulate several stacks.

Embedded functions for context and object reference management Embedded functions for objects and context management operate on stacks; see Table 2. These functions are executed in-line, that is, they are evaluated immediately after application of the rule that generated them in the string and before applying the next rule.

Syntactically, embedded functions are denoted by the stack name, followed by the function name, followed by an argument list (which may be empty):

`stack.function_name(`arg_1`, …, `arg_n`)`,

where *n* is a natural number (possibly zero). A function returns an object, an empty string, or a special object NIL.

Conditionals A conditional expression is used for making decisions during the rule processing. Its format is:

```
   if <condition1> then <string1>
elseif <condition2> then <string2>
...
else                     <stringn>
endif
```

where the parts `elseif` and `else` are optional. This in-line function returns $string_1$ if $condition_1$ is met, $string_2$ if $condition_2$ is met, etc.

An example of the use of both embedded functions and conditional markers is given in the next section.

6 *Example*

Due to space limitations, we can only present a simple example with a few DTG rules, see Figure 1. This example shows how an imperative sentence is translated into computer commands. In the lower part of the figure, the stages of transformation are shown along with the rules applied at each stage.

Grammar:

```
R1. A[C:ARTDET] B[C:SUST] -->
          if vs.first (name     = B) ≠ NIL then vs.first (name     = B)
       elseif vs.first (property = B) ≠ NIL then vs.first (property = B)
       elseif cs.first (name     = B) ≠ NIL then cs.first (name     = B)
       else                                      error
       endif
R2. B diferencia entre D y F ? --> diferencia D, F
R3. el  --> el{C:ARTDET, S:SM}
R4. la  --> el{C:ARTDET, S:SF}
R5. los --> el{C:ARTDET, S:PM}
R6. las --> el{C:ARTDET, S:PF}
R7. A[C:ARTDET] B[C:ADJ] --> A B[C:SUST]
R8. diferencia A[C:OBJ], B[C:OBJ] --> cs.push(A) cs.push(B) diff(A,B)
```

Sentence: *A ver, ¿cuál es la diferencia entre el tercero y el cuarto?*
'Let me see... what's the difference between the third and the fourth one?'

```
R2 gives: diferencia el tercero, el cuarto
R3 gives: diferencia el{C:ARTDET,S:SM} tercero, el{C:ARTDET,S:SM} cuarto
R7 gives: diferencia el{C:ARTDET,S:SM} tercero{C:SUST}, el{C:ARTDET,S:SM} cuarto{C:SUST}
R1 gives: diferencia objcat021_03 objcat021_04
R8 gives: diff(objcat021_03,objcat021_04)
   side effect: cs.push(objcat021_03), cs.push(objcat021_04)
```

Figure 1. Example of DTG Rules and sentence processing

It is supposed that the sentence in question is a query presented in the context (scene) of a catalog that shows numbered elements.

In this example, the system manipulates two stacks: a visual context stack `vs` and a conversation context stack `cs`. The visual context stack represents the objects in the common view of the system and the user, namely, the objects shown in the screen. This stack is maintained directly by the SOC system's engine and not by the grammar. The conversation context stack contains the objects mentioned during the dialogue and grows as the dialogue develops.

Rule **R1** is used for handling object reference; see Section 5. The object mentioned in the user's utterance is sought first in the common view; it can be referred either by name (*la silla* 'the chair') or by a property (*la roja* 'the red one'). If it is not found in the common view, then the most recently mentioned object with the given name is sought in the past conversation history.

Rule **R2** eliminates words that are not meaningful for the request, e.g. polite expressions.

Rules **R3** to **R6** state that *el*, *la*, *los*, *las* 'the' are forms of the definite article (corresponding in Spanish to different genders and numbers).

Rule **R7** uses wildcards to convert a combination of article and adjective into a noun, e.g., *el tercero* lit. 'the third' is converted into a substantive 'the third one'.

Rule **R8** is finally applied after the rule R1 has rewritten `tercero{C:SUST}` 'the third one' and `cuarto{C:SUST}` 'the fourth one' as references to specific objects, `objcat021_03` and `objcat021_04`. They are also added to the conversational context stack for future reference.

7 Conclusions

Spatial Object Composition (SOC) systems have characteristics that facilitate translating directly from natural language sentences into computer commands. Namely, in a SOC System, the language used is imperative; objects are previously defined and can be combined to create new ones; they have properties; they are always present; a spatial common representation exists visually; and a limited number of actions exist over these objects.

Given these characteristics, we have shown how such translation can be done with the Direct Translation Grammar. We have presented a framework based on this grammar and a mechanism for object reference and context management. The problem of resolving object references is solved within this framework

through a context stacks mechanism and conditionals embedded in the rules of the DTG.

The system can be extended to allow creating new rules out of existing ones. Then its capabilities can be dynamically extended through the dialogue with the user. This is a topic of our future work.

References

Asoh, Hideki, T. Matsui, J. Fry, F. Asano, and S. Hayamizu. 1999. A spoken dialog system for a mobile office robot, in *Proceedings of Eurospeech* '99, pp.1139-1142, Budapest.

Barros, Flavia de Almeida. 1995. *A Treatment of Anaphora in Portable Natural Language Front Ends to Data Bases*, PhD Thesis. University of Essex, UK.

Burton, Richard. 1992. Phrase-Structure Grammar. In Shapiro, Stuart ed., *Encyclopedia of Artificial Intelligence*. Vol. 1.

Chomsky, Noam. 1957. *Syntactic Structures*. Reprint Edition (1975), Mouton de Gruyter.

Craig, J., S. Berezner, C. Homer, and C. Longyear. 1966. DEACON: Direct English Access and Control. In *Proceedings of AFIPS Fall Joint Conference*, Vol 29, pp. 365-380, San Francisco, CA.

Di Eugenio, Barbara. 1993. *Understanding Natural Language Instructions: a Computational Approach to Purpose Clauses*. Ph.D. thesis, University of Pennsylvania, December. Technical Report MS-CIS-93-91.

Di Eugenio, Barbara. 1996. Pragmatic overloading in Natural Language instructions. *International Journal of Expert Systems* 9.

Fillmore, Charles. 1968. The case for case. In *Universals in Linguistic Theory*. Edited by Bach, Emmon and Harms, Robert T., 1-90. Chicago: Holt, Rinehart and Winston.

Gazdar, Gerald. 1982. *Phrase Structure Grammar*, in P. Jacobsen, and G.K. Pullum, eds., *The Nature of Syntactic Representation*, Reidel, Boston, MA.

Grosz, Barbara, D. Appelt, P. Martin, and F.C.N. Pereira. 1987. TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. In *Artificial Intelligence*, vol 32, pp 173-243.

Harris, Larry. 1984. Experience with INTELLECT: Artificial Intelligence Technology Transfer. In *The AI Magazine*, 2:2, pp 43-50.

Hendrix, Gary, E. Sacerdoti, D. Sagalowowicz, and J. Slocum. 1978. Developing a Natural Language Interface to Complex Data. In *ACM transactions on Database Systems*; 3:2, pp 105-147.

Joshi, Aravind. 1992. Phrase-Structure Grammar. In Shapiro, Stuart ed., *Encyclopedia of Artificial Intelligence*. Vol. 1. John Wiley & Sons, Inc. Publishers, New York.

Kay, Martin. 1979. Functional grammar. In *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*. 142-158.

Knight, Kevin. 1992. Unification. In Shapiro, Stuart ed., *Encyclopedia of Artificial Intelligence*. Vol. 2. John Wiley & Sons, Inc. Publishers, New York.

Pineda, Luis and G. Garza. 2000. A Model for Multimodal Reference Resolution. In *Computational Linguistics*, 26:2, pp. 139-193.

Pullum, Geoffrey. 1999. Generative Grammar, In Frank C. Keil and Robert A. Wilson (eds.), *The MIT Encyclopedia of the Cognitive Sciences*, pp. 340-343, Cambridge, MA, The MIT Press.

Sag, Ivan and T. Wasow, *Syntactic Theory: A Formal Introduction*. CSLI Publications, 1999.

Shinyama, Yusuke, T. Tokunaga and H. Tanaka. 2000. Kairai - Software Robots Understanding Natural Language. *Third Int. Workshop on Human-Computer Conversation*, Bellagio, Italy.

Watt, W., Habitability. 1968. *American Documentation* 19, pp. 338-351.

Webber, Bonnie. 1995. Instructing animated agents: Viewing language in behavioral terms. *Proceedings of the International Conference on Cooperative Multi-modal Communications*, Eindhoven, Netherlands.

Weischedel, Ralph. 1989. A Hybrid Approach to Representation in the JANUS Natural Language Processor. In *Proceedings of the* 27th *ACL*, Vancouver, pp 193-202.

Winograd, Terry. 1972. *Understanding Natural Language*. New York: Academic Press.

Winograd, Terry. 1983. *Language as a Cognitive Process. Volume I: Syntax.* Stanford University. Addison-Wesley Publishing Company.

Woods, William, R. Kaplan, and B.L. Nash-Webber. 1972. The Lunar Science Natural Language Information System: Final Report. BBN Report No. 2378. Bolt, Beranek and Newman Inc. Cambridge, MA.