Action-Request Dialogue Understanding System[*]

Hiram Calvo and Alexander Gelbukh

Center for Computing Research, National Polytechnic Institute,
Av. Juan de Dios Bátiz s/n, esq. Av. Mendizábal, México, D.F., 07738. México
hcalvo@alumnos.cic.ipn.mx, gelbukh@cic.ipn.mx, www.gelbukh.com

Abstract.

Human-machine communication in real world situations is usually a dialogue that consists of the user's utterances asking for information or requesting an action, followed by the computer's response answering or carrying out the requested task. In this paper we present a system that handles this kind of interaction through the use of a rewriting rules grammar with property modification, wildcard substitution and in-line functions. Our goal is to produce one or several specific computer instructions derived from the user's utterance. Objects referenced within the utterance are translated into symbols by the grammar rules and the use of special context objects called scenes, after which the instructions are executed by an external system. We present examples for the task of placing three-dimensional objects using instructions in Spanish language.

Keywords: Natural Language Processing, Rewriting Rules, Referenced Object Searching, Three-dimensional Objects Placing.

1 Introduction

An action-request system is an integrated system where a user interacts with a computer asking questions or requesting actions to be performed within a restricted subject area, while the computer answers or carries out the requested task. This kind of system has a specific domain and a limited number of objects and actions that can be performed over them. This limitation permits us to think of a natural language interface where actions requested and questions asked are posed to the system in a free and natural form. The system must react to this requests performing what it is asked to do in a coherent manner. By this way, the user can verify that the system has understood his or her query.

Since our objective is different from that of generative grammars, created to represent the well-formedness of sentences of a given language [1], we consider using a different formalism that permits a system to show its understanding of natural language by performing what it is asked to do. We propose using a rewriting rules grammar with property modification, wildcard substitution and in-line functions. The

[*] Work done under partial support of Mexican Government (CONACyT, SNI), PIFI and CGEPI-IPN, Mexico, and RITOS-2. We thank I.A. Bolshakov and G. Sidorov for useful comments.

purpose of this grammar would be to reduce language expressions to a logical form (a set of instructions) applying rewriting rules directly over utterances.

In order to illustrate how this system works, we assume a task where geometric objects such as spheres, tori, planes, and other basic geometric shapes can be combined in a three-dimensional canvas. This graphic task was chosen because it permits object representation to be shared among human and computer when it is visualized. This is an important characteristic that allows us to have a common context for the user and the system. Without this particular feature, referenced object searching would not be possible as it is presented in section 3.

In addition, a visual task like this permits us to create objects of higher complexity through Constructive Solid Geometry using operators like union, intersection, difference, and merge [3]. This way, new objects can be learned. Also, objects can have properties such as texture, reflectiveness, refraction index, etc. that we may modify through natural language.

One of the first systems that dealt with objects using natural language (an action-request system) was SHRDLU. In 1970, Terry Winograd wrote a program that could sustain a dialogue with a user about a small world of objects. It was written in MacLisp for the ITS system (an operative system for the PDP-10) and soon it became outdated because of periodical changes to the ITS. Nowadays, we do not know a system capable of handling the demonstration dialogue shown in [4]. However, our objective is not to resurrect SHRDLU, but to present a general system that can handle tasks involving space, objects and natural language instructions.

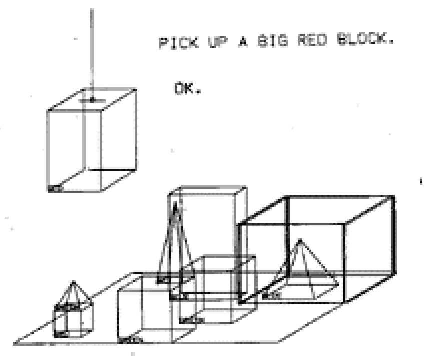


Figure 1. SHRDLU object world.

Examples presented throughout this paper and the grammar rules presented for processing them are oriented to Spanish language, although other languages could be supported.

As an example of the kind of utterances the user may pose to the system, consider the following request: *¿Me puedes poner el tercero junto al toroide?* 'Can you put the third one next to the torus?'. After applying 7 rewriting rules (detailed in section 4) over this utterance, we obtain the following instructions:

`move(objcat021_03, getpos(obj002)); sos.push(obj003); oos.push(obj002).`

The instruction `move(objcat021_03)` can be performed by an external function that moves the requested object. `sos.push(obj003)` and `oos.push(obj002)` are in-line functions used for object and context management. These functions are discussed in section 3.

In the next sections we will cover the grammar in detail, then object and context management, and lastly, an example of how are processed 4 utterances.

2 Rewriting Rules

In this paper, we propose the use of rewriting rules in the form $\alpha \rightarrow \beta$, where α and β are a string composed by one or more of the following elements (explained below):

- external procedures with arguments,
- a word or a phrase segment,
- a stem word with properties,
- wildcards with substitution possibility and property access,
- symbolic references to particular objects (symbols),
- in-line functions with arguments, and
- conditional markers.

Within the rule set no two rules with same left side (α) can coexist. The processing of rules has an order. Rules where α consists only of words or phrase segments are processed first, ordered by the length of α from longest to shortest. Then, the rest of the rules is processed ordered in the same way by the number of symbols that compose α.

Processing for each utterance stops when there are no more rules to apply. It is said that rules have been applied successfully over an utterance when it has been completely transformed to procedures, functions, and/or symbols separated by semicolons.

Below we describe in detail each one of the rule elements.

2.1 External procedures with arguments

External procedures with arguments are formed by a procedure name followed by arguments. Procedures do not return any type of information. They are executed by an external system after the successful application of rules over an utterance.
definition:

`procedure_name(`arg_1`, `arg_2`, ... , `arg_n`)`
n is a natural number, including 0

Example:
`move(A,B)` is an external procedure that places object `A` in position `B`.

2.2 Words or phrase segments

Words or phrase segments are strings of words considered together as they appear in dialogue. We consider neither their meaning nor their context. What is meaningful for us is that they appear as a pattern in exact order.

Example of rule containing a phrase segment at the left side:

`el catálogo de materiales metálicos → cat021` (1)

The rule (1) contains the string of words *el catálogo de materiales metálicos* 'the catalogue of metallic materials' but we are not analyzing their meaning. When this string of words appears together we can evoke (in the sense that [5] denotes) object `cat021`. This object has a meaning because it is an specific object and has specific properties.

2.3 Stem words with properties

Given a lexicon, we can recognize a stem that has been inflexed to make its meaning more specific. In addition, we are able to determine the grammatical category to which this word belongs. Within our rewriting rules, properties of a stem word appear surrounded by brackets {} indicating its abbreviated property name followed by a colon and its value. Stem words can have one or more of the properties and values shown in table 1.

Notation	Property	Values
C	category	SUST (noun), V (verb), ADJ (adjective), ADV (adverb), PRO (pronoun), ARTDET (definite article), ARTIND, (indefinite article), OBJ (object).
G	gender	M (masculine), F (feminine).
N	number	S (singular), P (plural)
T	verbal tense	PRES (present), INF (infinitive), IMP (imperative), SUBJ (subjunctive)
S	subject form	conformed by the subject number followed by its gender
O	object form	conformed by the object number followed by its gender
Q	quantity	LL, L, M, P, PP (equivalent to subjective appreciations mapped to arbitrary quantities understood as very little, little, medium, big, very big).

Table 1. Properties and Values for Stem Words.

Example of rule with properties for stem words (properties are surrounded by brackets):

`puedes    poder{C:V,T:PRES,S:2S}` (2)

This rule establishes that whenever the form *puedes* 'you are able' is found, it must be considered as an inflection of the verb *poder* 'to be able'. The properties for this word are category (verb), tense (present), and subject form (second person singular subject: *tú* 'you'). Note that `poder` is different to `poder{C:V,T:PRES,S:2S}`. The first one is a word while the second one is a stem word with properties. Specifying only `poder` in a rule does not refer to `poder` with properties.

2.4 Wildcards with substitution possibility or property access

To generalize a rule, one must be able to apply it over a set of words that share one or more properties. To do this, we use wildcards, which are a temporal reference (their scope is only the rule) that can be labeled as desired for future reference within the rule. To avoid confusion, it is advisable not to choose a common word to label a wildcard. If the wildcard will not be used at the right side of the rule, it can be left unlabeled. Wildcard properties appear within square brackets "[ ]".

If the wildcard at the left side of the rule has certain properties that are not matched by the current transformation, then the rule is not applied. On the contrary, if there are properties not stated explicitly by the wildcard, they will not be taken into account while matching.

Example:

```
[C:ARTDET] B[C:SUST]                                  (3)
```

This establishes the pattern as the definite article followed by a word that is a noun. Within the rule, this latter word can be referred as `B`. At the right side of the rule, properties can be modified using the same name of the temporal reference created at the left side. Properties that do not appear in this case will remain unchanged.

Example:

For the collocation *podrías juntarlo* 'could you approach it', in this case a polite use for imperative *júntalo* 'approach it', we apply the following rules:

```
podrías → poder{C:V, T:SUBJ, S:2S}                    (4)
juntarlo → juntar{C:V, T:INF, O:3SM}                  (5)
```

then, by combining (4) and (5) using rule (6):

```
poder{C:V,T:SUBJ,S:2S} A[C:V,T:INF] → A[T:IMP]        (6)
```

we transform the original collocation into (7):

```
juntar{C:V,T:IMP,O:3SM}                               (7)
```

Due to wildcards, this scheme works also for every polite use of *podrías* + 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 properties for other objects outside brackets is possible through use of dot notation:

Consider the following example:

```
juntar{C:V,T:IMP,O:3SM} un poco más
```

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

```
un poco más → {C:ADV, Q:L}                            (8)
```

then, using another rule, we can apply it to the verb:

$$\texttt{E[C:V] D[C:ADV,Q]} \rightarrow \texttt{E[Q:D.Q]} \qquad (9)$$

Rule (9) can be read as: "Let `E` be a word, whose category is a verb, followed by `D`, an adverb of quantity. Transform `E`, so that its property `Quantity` is the same as the one of `D`".

When a property is specified within a wildcard, but its value is not specified, it indicates that this wildcard requires the property to be present for matching, regardless of its value.

2.5 Symbolic references to particular objects

A scene is an object composed by other objects. In turn, each one of these objects can be made of other objects. Objects can have properties, but not the ones that are used for stem words. An object can have one of the following properties: position, size, components, color, material, density, alterability, shape, and a set of actions that can be applied to this object. When we talk about "object", we are talking of the symbolic reference of a particular referent [6]. Catalogues are objects too, composed by elements that are objects as well. It is important to remark here that catalogue objects belong to one context (the catalogue), while objects in the canvas (for the task of geometric object design) belong to another context. Each one of these contexts is called a scene. The role of context in successful interpretation will be discussed in section 3.

Example:

`obj002` is a particular sphere that appears in the scene (say, the blue one at the corner).

2.6 In-line functions with arguments.

In-line functions with arguments are formed by a function name followed by arguments. A function must return an object or a value. Functions with arguments are executed when each rule is applied.

definition:
`object function_name(arg`$_1$`, arg`$_2$`, ... , arg`$_n$`)`
n is a natural number, including 0
where `n` is a natural number, including 0
Example:

`ss.last(name=obj462)`	returns true if the obj462 exists in the current scene (ss) or false if it does not. (more on scenes in section 3)

2.6.1 Conditional markers

A conditional marker is a function used for taking decisions within the rule processing. Its format is: `if(<condition>,<object₁>,<object₂>)`. This in-line function returns object$_1$ if <condition> is met or <object$_2$> if it is not.

Example:

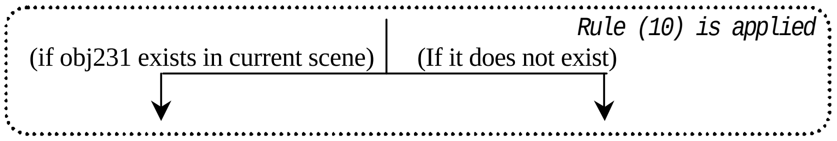
For the utterance *Juan mira al hombre con un telescopio* 'John looks the man with a telescope', the system must decide whether to apply a rule or another considering if *el hombre con un telescopio* exists in the current context (scene) using the `last` function. (If the object does not exist, `last` returns false). The rule for doing this is:

```
el hombre con un telescopio → if(ss.last(name=obj231),obj231,
                                 el hombre con un telescopio)    (10)
```

If the object `obj231` does not exist in the current scene, *el hombre con un telescopio* is replaced by *el hombre con un telescopio*. That is, it is not changed.

The next table shows the steps of transforming the original utterance into symbols. Other rules than (10) are not specified for this example. The underlined elements are those that have become a symbol.

```
Juan mira al hombre con un telescopio
Juan mira a el hombre con un telescopio
```



$\underline{\texttt{obj121}}$ `mira a` $\underline{\texttt{obj231}}$	`Juan mira a el hombre con un telescopio`
$\underline{\texttt{obj121}}$ $\underline{\texttt{look\{C:V\}}}$ $\underline{\texttt{obj231}}$	`Juan mira a` $\underline{\texttt{obj230}}$ `con un telescopio`
	`Juan mira con un telescopio a` $\underline{\texttt{obj123}}$
	`Juan mira con` $\underline{\texttt{obj413}}$ `a` $\underline{\texttt{obj123}}$
	$\underline{\texttt{obj121}}$ $\underline{\texttt{looks\{C:V,W:obj413\}}}$ $\underline{\texttt{obj123}}$

Table 2. Example of derivation using in-line conditional markers

Another example of the `if` function is shown in section 4.1

3 Object and context management

Each time a we reach a noun, a pronoun, or a noun phrase, we must find a unique symbolic reference to a particular object. However, the same expression (string of words) can be used to reference several particular objects depending on context. In order to transform an expression to a symbolic reference we must first determine a context for it [7]. To provide context handling, we consider context as an object that contains objects within. It is called a scene object. Context and object handling is supported by three stacks: subject-object stack (`sos`), object-object stack (`oos`) and context or scene stack (`ss`). A context change will occur when we shift our attention from the object itself into its components. For example, we could think of parts or elements of a catalogue.

Besides standard operations over stacks (push, pop), we can search for objects by property. Functions for objects and context management are listed below. These functions are executed in-line (that is, they are evaluated before applying another rule). Parameters surrounded by brackets are optional.

`push(<object>)`	adds <object> to the top of the stack
`object pop()`	extracts <object> from the top of the stack
`object last({<property> = <value>})`	when it is called without parameters, it returns <object> from the top of the stack without popping it. If a condition is established, then it searches for the first object that meets this condition beginning from the top of the stack. If no object is found, it returns false.

Using the three stacks we can define the procedure for referenced object searching:

```
search object in sos
   if it is not found: search object in oos,
       if it is not found: search object in ss,
          if it is not found: until it is found do:           (P1)
                scene.pop() and search object in ss.
```

This procedure can be embedded as a rule. This is shown in section 4.1.

4 Processing of Sample Queries

Here we present a rule set that is able to process several utterances in Spanish language. These utterances (abridged utt) are inspired by the dialogue presented in [8].

utt.1: *¿Me puedes mostrar el catálogo?* 'Can you show me the catalogue?'
utt.2: *¿Me puedes mostrar el catálogo de objetos formados por esferas?* 'Can you show me the catalogue of objects made of spheres?'
utt.3: *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?'
utt.4: *¿Me puedes poner el tercero junto al toroide?* 'Can you put for me the third one next to the torus?'

4.1 Rules

Here is the set of rules used for analysis of the fragment of utterances presented above. Rule 1 synthesizes procedure P1 for referenced object searching.

```
1   A[C:ARTDET] B[C:SUST] ->
    if(sos.last(name = A.name),sos.last(name = A.name),
    if(oos.last(name = A.name),oos.last(name = A.name),
    if( ss.last(name = A.name), ss.last(name = A.name),
    ss.pop();A B))))
2   [C:ARTDET] cuarto{C:ADJ} -> 4{C:SUST}
3   [C:ARTDET] tercero{C:ADJ} -> 3{C:SUST}
4   B diferencia entre D y F -> diferencia D F
5   B[C:SUST,Q] -> if (ss.last(name = B.name),ss.last(name = B.name),
           ss.last(prop = B.Q))
6   el -> el{C:ARTDET,S:SM}
7   nextto A[C:OBJ] -> getpos(A); oos.push(A)
8   cuarto -> cuarto{C:ADJ,S:SM}
9   diferencia A[C:OBJ] B[C:OBJ] -> diff(A,B);sos.push(A);sos.push(B);
10  el catálogo de objetos formados por esferas -> cat021
11  junto a -> nextto
12  me poder{C:V,T:PRES,S:2S} A[C:V,T:INF] -> A[T:IMP,S:2S,O:1S]
13  mostrar -> mostrar{C:V, T:INF}
14  mostrar{C:V,T:IMP,S:2s} A[C:OBJ] -> show(A); ss.push(A);
15  poner{C:V,T:IMP,S:1S} B[C:OBJ] D[C:POS] -> F=move(B,D);sos.push(F);
16  puedes -> poder{C:V,T:PRES,S:2S}
17  tercero -> tercero{C:ADJ,S:SM,Q:3}
18  catálogo -> catálogo{C:SUST,S:SM,Q:4}
19  poner -> poner{C:V,T:INF,S:1S}
20  al -> a el
```

4.2 Rules in action

Now we can process the utterances presented at the beginning of this section. The number at the left indicates the number of the rule used between one step and the next.

utt1: ¿Me puedes mostrar el catálogo? ‘Can you show me the catalogue?’

```
16. me poder{C:V,T:PRES,S:2S} mostrar el catálogo
13. me poder{C:V,T:PRES,S:2S} mostrar{C:V,T:INF} el catálogo
12. mostrar{C:V,T:IMP,S:2S,O:1S} el catálogo
18. mostrar{C:V,T:IMP,S:2S,O:1S} el catálogo{c:SUST,S:SM}
 6. mostrar{C:V,T:IMP,S:2S,O:1S} el{C:ARTDET,S:SM} catálogo{c:SUST,S:SM}
 1. mostrar{C:V,T:IMP,S:2S,O:1S} objcat01
14. show(objcat01);ss.push(objcat01);
```

utt2: ¿Me puedes mostrar el catálogo de objetos formados por esferas? 'Can you show me the catalogue of objects made of spheres?'

```
16. me poder{C:V,T:PRES,S:2S} mostrar el catálogo de objetos formados
    por esferas
13. me poder{C:V,T:PRES,S:2S} mostrar{C:V,T:INF} el catálogo de objetos
    formados por esferas
12. mostrar{C:V,T:IMP,S:2S,O:1S} el catálogo de objetos formados por
    esferas
10. mostrar{C:V,T:IMP,S:2S,O:1S} cat021
14. show(cat021);ss.push(cat021);
```

utt3: 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?'

```
 4. diferencia el tercero el cuarto
 6. diferencia el{C:ARTDET,S:SM} tercero el cuarto
 6. diferencia el{C:ARTDET,S:SM} tercero el{C:ARTDET,S:SM} cuarto
 2. diferencia el{C:ARTDET,S:SM} tercero 4{C:SUST}
 3. diferencia 3{C:SUST} 4{C:SUST}
 5. diferencia objcat021_03 4{C:SUST}
 5. diferencia objcat021_03 objcat021_04
 9. diff(objcat021_03,objcat021_04); sos.push(objcat021_03);
    sos.push(objcat021_04);
```

utt4: ¿Me puedes poner el tercer junto al toroide? 'Can you put for me the third one next to the torus?'

```
16. me poder{C:V,T:PRES,S:2S} poner el tercero junto al toroide
19. me poder{C:V,T:PRES,S:2S} poner{C:V,T:INF,S:1S} el tercero junto al
    toroide
 6. me poder{C:V,T:PRES,S:2S} poner{C:V,T:INF,S:1S} el{C:ARTDET,S:SM}
    tercero junto al toroide
20. me poder{C:V,T:PRES,S:2S} poner{C:V,T:INF,S:1S} el{C:ARTDET,S:SM}
    tercero junto a el toroide
 6. me poder{C:V,T:PRES,S:2S} poner{C:V,T:INF,S:1S} el{C:ARTDET,S:SM}
    tercero junto a el{C:ARTDET,S:SM} toroide
11. me poder{C:V,T:PRES,S:2S} poner{C:V,T:INF,S:1S} el{C:ARTDET,S:SM}
    tercero nextto el{C:ARTDET,S:SM} toroide
17. me poder{C:V,T:PRES,S:2S} poner{C:V,T:INF,S:1S} el{C:ARTDET,S:SM}
    tercero{C:ADJ,S:SM} nextto el{C:ARTDET,S:SM} toroide
12. poner{C:V,T:IMP,S2s,O1S} el{C:ARTDET,S:SM} tercero{C:ADJ,S:SM}
    nextto el{C:ARTDET,S:SM} toroide
 3. poner{C:V,T:IMP,S2s,O1S} 3{C:SUST} nextto el{C:ARTDET,S:SM} toroide
 5. poner{C:V,T:IMP,S2s,O1S} objcat021_03 nextto el{C:ARTDET,S:SM}
    toroide
 1. poner{C:V,T:IMP,S2s,O1S} objcat021_03 nextto obj002;
    oos.push(obj002);
 7. poner{C:V,T:IMP,S2s,O1S} objcat021_03 getpos(obj002);
    oos.push(obj002);
15. move(objcat021_03,getpos(obj002));sos.push(obj003);
    oos.push(obj002);
```

In the last line, the object is copied when it is moved from one context to another.

5 Conclusions and future work

In this paper, we have presented a system that can derive one or more specific computer instructions from a request by the end-user. Objects referenced within this request are translated to symbols by a rewriting rules grammar with property modification, wildcard substitution, in-line functions, and the use of special context objects called Scenes. Instructions are executed then by an external system.

The system can be used for computer tasks that meet the following conditions: the user and the computer share common contexts, which are visualized, and the application domain is limited. We have presented examples for the task of placing objects in three-dimensional canvas. This work can be extended to cover not only abstract geometric objects, but also world objects within a modeled world. In addition, by using special functions this system can be enhanced to allow new rules to be created by previous rules, thus, automating dynamic grammar extension through dialogue.

6 References

[1] Ivan Sag, and Thomas Wasow. *Syntactic Theory: A Formal Introduction*. CSLI Publications, 1999, Appendix B.

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

[3] R.A. Goldstein, & R. Nagel. *3-D Visual Simulation*. Simulation 16, 1971, pp. 25-31

[4] Terry Winograd, *Understanding Natural Language*. New York: Academic Press, 1972 cited in [2] and in http://hci.stanford.edu/~winograd/shrdlu.

[5] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000. p. 672.

[6] C. K. Ogden and I. A. Richards. *The Meaning of Meaning*, 8th ed. (1923; New York: Harcourt Brace Jovanovich, 1946), pp.9-12. cited on J.F. Sowa, *Conceptual Structures. Information Processing in Mind and Machine*. Addison Wesley, 1984, p.11

[7] L. A. Pineda, G. Garza. A *Model for Multimodal Reference Resolution*. Computational Linguistics, Vol. 26, No. 2., 2000, pp. 139-193

[8] L. A. Pineda, A. Massé, I. Meza, M. Salas, E. Schwarz, E. Uraga, L. Villaseñor. *The DIME Project.* Department of Computer Science, IIMAS, UNAM, 2002.