

# **Human-Robot Interaction Manual**

## **Using the Natural Language Components**

Dennis Perzanowski  
Code 5512  
Naval Research Laboratory  
Washington, DC 20375

Myriam Abramson  
ITT Industries  
Alexandria, VA 22303

Voice mail: 202-767-9005  
Fax: 202-767-3172  
E-mail: <dennisp | abramson> @aic.nrl.navy.mil

### **1. Introduction**

The natural language component of the interface is made up of three sub-modules: ViaVoice Speech Recognition, which maps the acoustic input into a text string, and Nautilus, which translates the text string into a logical form. Tranfuns takes this logical form, plus any output from the gesture recognition system, and maps it into a robot command, which is also in the form of a text string. The output is given to the Task Manager for subsequent processing by the various robot modules to produce the corresponding robot behaviors. An overview showing the mapping from voice input to robot command is given in Figure 1.

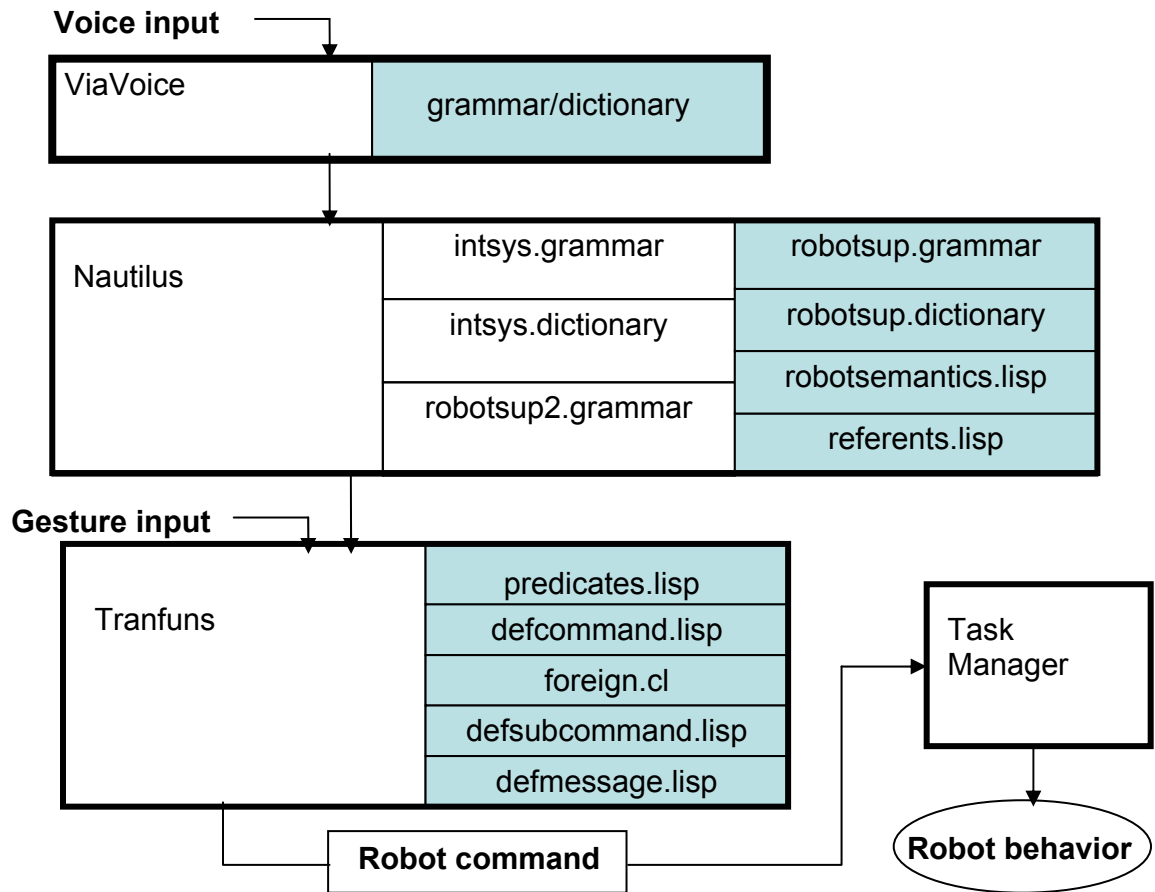


Figure 1. Overview of Natural Language and Gesture Processing

The shaded areas are those that will need changes for adding to the range of utterances. A detailed discussion follows.

ViaVoice is a commercial off-the-shelf product. We currently are using ViaVoice Millennium Edition on PCs running Windows 98 and XP, and we are currently moving to Linux. ViaVoice requires a compiled grammar/dictionary using IBM's ViaVoice SDK. We currently are using the SDK for Windows. While both products are available through IBM (educational pricing might be available), ViaVoice for Linux is also available, but I do not know of a comparable SDK for Linux. There is an SDK for Java Technology. To create a speech recognition grammar/dictionary for the Robot Challenge, both products (both ViaVoice and a comparable ViaVoice SDK) are required.

Nautilus is an in-house, Naval Research Laboratory (NRL), natural language understanding system, and as such is provided free of charge.

## 2. Creating a Speech Grammar/Dictionary

The human-robot interface, Interbot, uses two subsets of grammars and dictionaries. For speech recognition, ViaVoice is used. When adding to the range of utterances for new or expanded applications, ViaVoice may require new non-terminal expansions and terminals in the *foo.bnf* file to be compiled, and new pronunciations may have to be

added if the ViaVoice library does not already contain the pronunciations for terminals in the new or expanded application.

ViaVoice requires a compiled grammar/dictionary. We will discuss the creation of the grammar and dictionary for understanding in the next section. An example of the speech recognition (ViaVoice) grammar/dictionary follows in Figure 2:

```

<root> =                <robot-name> <robot-command> | <robot-name>
                        <robot-command> |
                        <robot-name> <robot-yesnoq> |
                        <robot-name> <robot-whquestion> .

<robot-whquestion> =    where is the <objects-in-world> |
                        where are the <plural-objects-in-world> |
                        ...
                        how many objects do you see |
                        what (object | objects)? do you see |
                        what (object | objects)? do you see <loc12> you | ...

<objects-in-world> =    chair | table | pillar | box | computer .

<plural-objects-in-world> = chairs | tables | pillars | boxes |
                           computers .

<loc12> =                behind | in front of .

```

Figure 2. Example of ViaVoice grammar/dictionary file, *foo.bnf*

Dictionary items are entered directly into the grammar/dictionary file, *foo.bnf*, as either literals, as in the expansion of **<loc12>** in Figure 2, or generalizations can first be captured, such as in generalizing that wh-questions can be uttered about objects or things, **<objects-in-world>**, which further expands into the literals, **chair**, **table**, etc.

Thus, the speech grammar is defined by enumerating the valid words and phrases of the domain. Each production rule is basically of the form: **<rule> = sentences and phrases** .

The left-side of each rule must be unique. The assignment operator ( = ) and the terminal punctuation ( . ) are required. Sentences and phrases can be either terminal, e.g. foo, or nonterminal symbols <foo>. A vertical bar ( | ) indicates mutually exclusive expansions of the left-side of the production. A question mark ( ? ) after either terminal or nonterminal nodes indicates zero or one occurrence of the item. Other repetitive operators that can be used are the plus sign ( + ) to indicate one or more occurrences of the item, and the asterisk ( \* ) for zero or more occurrences of the item. Parens ( ( . . . ) ) indicate an either/or relationship of the items.

Using the grammar/dictionary fragment in Figure 1, the following subset of text strings could be generated: “coyote where is the pillar”, “coyote what do you see”, and “coyote what objects do you see in front of you”. We disregard punctuation in the generated text strings, such as capitalization and the use of commas and question marks, since it is unnecessary for further processing in the interface.

Having created a set of production rules for the ViaVoice grammar/dictionary, containing terminal and non-terminal nodes, and an incorporated dictionary, the grammar/dictionary, *foo.bnf*, must be compiled using the SDK.

The command-line instruction for compiling the grammar/dictionary is:

**vtbnfc -en -m -s foo.bnf .**

The executable file, *vtbnfc.exe*, should be in the same directory as *foo.bnf* to make compilation easier. **-en** flag indicates that there are no translations in the grammar/dictionary (cf. SDK documentation for a more detailed description of translations). **-m** flag instructs the compiler to permit mumble words; **-s** flag permits silences in the bnf expansions of the grammar.

At compilation time, ViaVoice checks pronunciations of terminal nodes in the grammar/dictionary against its library of pronunciations. Many common words are already cataloged and user-specific pronunciations will not be required; however, acronyms are a real problem (cf. the SDK documentation), so it is best to avoid them at this time. At compile time, if some pronunciations are not found, the user has to create pronunciations using the dictionary building tool, which is part of the SDK. Instructions for adding pronunciations can be found in the SAPI Developer's Guide, provided with the SDK.

Once all pronunciations are found and compilation is successful, the resulting *foo.fsg* file is used in the interface for speech recognition and mapping acoustic input into a text string for Nautilus to process.

### 3. Creating a Text Dictionary and Grammar

After ViaVoice converts the acoustic input of the speech signal to a text string, Nautilus uses its own dictionaries and grammars to parse, interpret and translate the text string to a usable robot command for the other modules of the system.

Nautilus has two basic dictionaries and two basic grammars. The first dictionary, *intsys.dictionary*, is an application non-specific dictionary. It contains many ordinary words, usually non-domain oriented words. **N.B. It should not be altered.** Instead, *robotsup.dictionary* is the file used to add new dictionary items to the application.

Dictionary items can be entered using three macros: **noun**, **verb**, or **word**. An example of the **noun** and **verb** macros in *robotsup.dictionary* follows in Figure 3.

```
(noun :root label :attributes (ncount))
```

```
(verb :root label :objlist (NSTGO NN))
```

Figure 3: **Noun** and **verb** macros in supplementary dictionary

The **noun** macro in Nautilus takes two arguments, **:root** and **:attributes**; the **verb** macro takes two arguments, **:root** and **:objlist**. In Figure 3, *label* in English is both a noun and a verb; therefore, it is listed as both in the dictionary. The so-called root of the noun and verb is regular; therefore, Nautilus knows what the plural of the noun is (*labels*), and it also knows the principle parts of the verb: (3<sup>rd</sup> person singular = *labels*; past tense = *labeled*; present participle = *labeling*; past participle = *labeled*) based upon **:root**.

**:attributes** is a list of special lexical features, such as the “countability” (=ncount) of the English word *label*, e.g. “one label,” “two labels,” etc. These attributes are listed toward the top of the two grammar files *intsys.grammar* and *robotsup.grammar*. They are fairly straightforward.

The **verb** macro takes **:root** but also an **:objlist** argument. This argument lists the various types of syntactic objects the verb can take. In Figure 2, *label* can have either an **NSTGO** (Noun **STrinG** Object) or an **NN** (Noun string + Noun string) object. Thus, in English, either “label the pillar,” (an example of **NSTGO** which typifies a single noun phrase as an object) or “label object number 1 a pillar” (an example of **NN** which typifies two noun phrases, [object number 1] and [a pillar], as an object) are acceptable utterances. Users can get a better feel for what kinds of objects are permitted after verbs, by looking at the expansion of **OBJECT** in both *intsys.grammar* and *robotsup.grammar*. These categories and attributes mentioned above are largely based on the work of Naomi Sager and The Linguistic String Project of the Courant Institute of New York University (Naomi Sager. 1981. *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley: Reading, MA). The examples given here are meant only as illustration.

Irregularities in either nouns or verbs must be specified, as in Figure 3.

```
(verb :root run :past ran :prespart running :objlist (ND NSTGO NULLOBJ
DSTG))

(word (look at) (V (SINGULAR xn (look-at) OBJLIST (NSTGO)))
TV (PLURAL xn (look-at) OBJLIST (NSTGO))))

(word this (T (xn THIS SINGULAR DEFINITE DEM)))
```

Figure 4. Irregularities and the **word** macro

Irregularities in the English verb *run* are specified, as in Figure 4. **V** and **TV** are grammatical categories of Verb and Tensed Verb, used in syntactic parsing. The **xn** attribute is a regularization of the lexical item that Nautilus uses in the syntactic parsing of these items. These regularities are carried over into the semantics and permit interpretation to proceed. The **:objlist** of the verb *run* defines the acceptable object strings for the verb: **ND** (Noun + **ADverb**, as in “run home quickly”), **NSTGO** (explained above), **NULLOBJ**, (no object, as in the command “run”), and **DSTG** (**ADverb STrinG**, as in “run quickly”).

There is a **word** macro which allows users to add unusual expressions, such as *look at* in Figure 4, which we treat as an idiom, because it is more easily interpreted as a single unit item. Finally, the demonstrative article **T**, *this*, in Figure 4 is tokenized by using the **word** macro.

Summarizing, new lexical items can be added to the application by entering new items in *robotsup.dictionary*. This and other necessary dictionaries are loaded at run-

time. In the Windows OS, the *InterRob.bat* file specifies the various environment variables and files to be loaded for the parsing of spoken utterances.

Changes to the Nautilus grammar(s) should be avoided. **N.B. In no case, should intsys.grammar or intsys.dictionary be changed.** Additions, deletions, modifications to *robotsup.grammar* may require troubleshooting, and if alterations to *robotsup.grammar* are made, it must be re-compiled. The compiled forms of the grammars and dictionaries are the \*.*gram-lisp* files. (There is also an additional pair of grammar files, *robotsup2.grammar/robotsup2.gram-lisp*, which **should not be changed** but must be loaded to accommodate idiomatic expressions in text strings. *InterRob.bat* and the files loaded by this file ensure that all files are loaded.)

Currently, we use a tool, DNautilus (Development Nautilus) to troubleshoot and compile Nautilus grammars. If only dictionary items are added or modified, no re-compilation of the grammar is required. We hope that the grammars supplied are robust enough to handle a wide range of syntactic constructions, so work on the Nautilus grammar will not be necessary. We should also mention that at this time, coordination, as in “go to the door *and* turn left,” or “look for the elevator *and* the sign,” are not supported. Only simple sentences are handled in both ViaVoice and Nautilus.

We anticipate you will only be making changes to *robotsup.dictionary*; however, we will supply direct support in the event that Nautilus grammar support is required.

#### 4. Making Semantic Changes and Mapping to Robot Commands

The addition of lexical items will necessitate additions (hopefully no alterations of existing semantic entities and rules) to the *robotsemantics.lisp* and *referents.lisp* files.

After Nautilus syntactically parses and regularizes the text string output by ViaVoice, it then gives each syntactic regularization a semantic interpretation which is further interpreted for such entities as objects in the world, etc.

*Predicates.lisp* specifies a special subset of predicates mapped to functions required for the special “verb” be in English. The file will probably not need alteration, but we list it here for completeness of coverage.

The basic structure of the semantics is based on a **defframe** macro. **Defframes** are semantic classes of verbs or nouns. They specify what semantic generalizations are captured in the **defpreds** and **defnpreds**, the latter being the specific verbs and nouns of the domain.

For example, Nautilus will regularize the utterance “*coyote go over there*” (punctuation is superfluous) to the representation (1). [This utterance requires a gesture to be understood. The Gesture Recognition is handled in another component. If a gesture is perceived, then the following mapping will proceed. If, on the other hand, no gesture is perceived, an appropriate response of “*Where?*” will be generated in a similar fashion to this example.]

```
(1) (ADDRESS (NAME N5 (:CLASS SYSTEM) COYOTE)
      (IMPER #:V7308 (:CLASS GESTURE-GO)
        (:AGENT (PRON N7 (:CLASS SYSTEM) YOU))
        (:GOAL (NAME N6 (:CLASS THERE) THERE))))
```

The file *referents.lisp* must be altered to accommodate any new objects added to the domain. The template for creating referents is the **makeobj** macro, listed at the top of the file and given here in (2):

```
(2) (MAKEOBJ OBJNAME CLASSES SPEC-WORDS (:ROLE {CLASSES |
      NAMES}))* )
```

where OBJNAME is the name of an entity;  
 CLASSES is the semantic class (one or more);  
 SPEC-WORDS is one or more of either (1) pronoun specifier (me, you) or  
 (2) permissible left noun modifiers;  
 ROLE is a TINSEL role followed either by one or more semantic classes  
 or entity names (strings);  
 \* specifies that multiple roles can be assigned.

Thus, *Coyote* and *you* are represented by the following makeobj macro (3) in this domain.

```
(3) (makeobj coyote system (you yourself coyote))
```

The semantic rules in *robotsemantics.lisp* (cf. Figure 5) enable Nautilus to obtain the interpretation in (1).

```
(defpred go (gesture-go :agent (S) :goal (LOC) :to-loc (TO THROUGH LOC O
      &)))
(defframe gesture-go
  :isa action :agent (system) :goal (goal-direction) :to-loc (thing gesture))
(defframe there :isa goal-direction)
```

Figure 5. Semantic rules for interpretation in *robotsemantics.lisp*

Nautilus maps the verb (semantically the **defpred**) *go*, of the sentence “*Coyote, go over there*” into the predicate **gesture-go**. The predicate takes a **Subject**, a **LOCative** goal, and optionally (&) a **:to-loc** argument which can be indicated by the prepositions **TO** or **THROUGH** or be a **LOCative** expression or **Object**. As a member of the semantic class of predicates called **gesture-go** verbs, **go** is an **action**, and all actions in this domain will further map to **defcommands** in the *defcommand.lisp* file.

The **defcommand** macro encapsulates the processing of gesture and speech by calling the appropriate functions for gestural commands. This macro is used to define the different rules of action disambiguating a spoken command. The rules are ordered by specificity according to the binding of the arguments of the macro and the conditions in the rule.

The **gesture-go** frame specifies the semantic class of arguments that can fill the argument slots of the predicate *go*. Given that the semantics knows that entities that are directly addressed, such as *Coyote*, are **NAMES**, the sentence “*Coyote, go over there.*” is interpreted.

The various gestures, or lack thereof, are linked to the domain predicate **gesture-go** in (1) by means of the **defcommand gesture-go** (cf. Figure 6). [For ease of exposition, specific parts of the rules under discussion are highlighted.]

```
(defcommand gesture-go (goal to-loc through-loc manner)
  (((no-gesture-p) (talk-and-echo "Where?"))
   ((misunderstood-gesture-p) (talk-and-echo "I'm sorry. I don't understand
    the gesture."))
   ((one-hand-gesture-p) (talk-and-echo "I'm sorry. I didn't see you gesture in
    any direction."))
   ((two-vectors-gesture-p) (talk-and-echo "I'm confused. You pointed in two
    different directions."))
   ((vague-gesture-p) (move-robot-with-direction goal to-loc)) ; direction
    only
   ((distance-gesture-p) (move-robot-to-distance goal to-loc))
   ((palm-gesture-p) (move-robot-with-palm-gesture goal to-loc)))
```

Figure 6. The **defcommand gesture-go**

If, for example, an appropriate gesture does not co-occur (***no-gesture-p***) with the utterance, the robot says “*Where?*”

If an appropriate gesture (***vague-gesture-p***) co-occurs with the utterance “*Coyote, go over there,*” the **defsubcommand move-robot-with-direction** of *defsubcommand.lisp* (cf. Figure 7) is called.

```
(defsubcommand move-robot-with-direction (goal to-loc)
  ((has-type to-loc 'door-gesture)
   (GoToDoor (location-coords)))
  (t (talk-and-echo (format nil "Going ~A" (format-gesture)))
    (GoObstacle (get-direction))))
```

Figure 7. The **defsubcommand move-robot-with-direction**

The **defsubcommand** macros called by **defcommands**, are ordered only by the conditions of the rule.

The **defmessage GoObstacle** (cf. Figure 8) called by the **defsubcommand move-robot-with-direction**, is given in *defmessage.lisp*. The **defmessage** macro sends an appropriate robot command to the robot for action.

```
(defmessage GoObstacle (direction)
  (send-token (format nil "4 ~D obstacle" direction)))
```

Figure 8. The **defmessage GoObstacle**



Thus, if an appropriate gesture is perceived, the message “4 ~**D obstacle**” (where ~**D** will have been replaced by an appropriate **direction**) will be sent to the Task Manager for robot action.

*Foreign.cl* lists the various foreign functions mapping the calls required by the C code in the robotics application to corresponding Lisp functions. Because information needs to get to the robotics modules for action to occur based on the utterance “*Coyote, go over there,*” a foreign function **send-token** (cf. Figure 9) maps the Lisp function to the corresponding C function **RF\_send\_token**.

```
(ff:def-foreign-call
  (send-token "RF_send_token")
  ((tokstr (* :char)))
  :strings-convert t
  :returning :int)
```

Figure 9. The foreign function **send-token**

If additional functionality is added on the “C-side” of the application, foreign functions will need to be added on the “Lisp-side.”

## 5. An Illustration

Let us look at an example where changes are required to the various files. Let’s use the sentence “Coyote, go over there” as our sample sentence and assume that the words *go*, *over*, and *there* are not in the grammars or dictionaries.

First, add a definition for the utterance “go over there” for ViaVoice in *foo.bnf*. Compile it.

Second, using the macros for dictionary entries, add dictionary items for all three words in *robotsup.dictionary*.

Third, check to make sure that the sentence now syntactically parses. If not, add an appropriate BNF and constrain using syntactic restrictions as illustrated in *robotsup.grammar*. Compile the new supplementary grammar; reload and retest until a satisfactory syntactic parse is obtained.

Fourth, add appropriate semantic representations for each of the words in *robotsemantics.lisp*. (If nouns are involved, *referents.lisp* will also need to be altered.)

Fifth, add appropriate commands and, if necessary, subcommands to *defcommand.lisp* and *defsubcommand.lisp*.

Sixth, add appropriate foreign functions in *foreign.lisp*, if necessary, so that an appropriate message is sent to the robot application.

## 6. Conclusion

Summarizing, Nautilus takes the ViaVoice text string, parses it syntactically, regularizes it and then submits the regularization for interpretation. After interpretation, the resulting representation is mapped to an appropriate robot command in *Tranfuns* and passed to the Task Manager. If the resulting representation is missing required information, such as an appropriate gesture, *Tranfuns* ensures that appropriate verbal error messages are output for further human interaction.

## **7. Acknowledgments**

The authors would like to express their appreciation to Alan Schultz, Bill Adams, and Magda Bugajska for their help in editing various drafts of this manual. Any errors, however, are the responsibility of the authors.

## **8. References**

- Grishman, R., (May 1990), "PROTEUS Parser Reference Manual," Project Report 00-4-C, New York University.
- Perzanowski, D., Schultz, A.C., Adams, W., Marsh, E., and Bugajska, M., (January/February 2001), "Building a Multimodal Human-Robot Interface," *IEEE Intelligent Systems*, vol. 16, no. 1, IEEE Computer Society, pp. 16-21.
- Wauchope, K., (1994), *Eucalyptus: Integrating Natural Language Input with a Graphical User Interface*, Technical Report NRL/FR/5510-94-9711, Naval Research Laboratory, Washington, DC.