Functional Descriptions as a Formalism for Linguistic Knowledge Representation in a Generation Oriented Approach

Miyo Otani [1] & Nathalie Simonin [2]
(Cap Sogeti Innovation, Paris, France)

Topic: Knowledge Representation and Use

Abstract:

This paper describes how linguistic knowledge has been declaratively formalised using Functional Descriptions (FDs), in the generation module of the SAGE system. SAGE (Sentence Analysis and GEneration) is the Natural Language Frontend of the Dialogue Manager of the Esprit I project Esteam-316.

The present implementation has the advantage of being based on two principles, which are the dynamic checking of constraints and the functional unification of knowledge and dynamic objects. In order to provide these functionalities to the former formalism of FD, we have introduce the notions of Syntactic Components *and of* Current Syntactic Component. *The whole sentence is built step by step in a complex tree-like structure. The generation interpreter is able to move upward and downward inside this tree.*

In addition, our system validates the use of a Lexicon-Grammar (drawn from the LADL studies) for sentence-generation. The target language is English, but all of the knowledge bases have been developed in such a way that the generation process is able to support a change of language.

Cap Sogeti Innovation
118, rue de Tocqueville
75017 Paris, France

tel: (33) -1- *46* 22 60 27
fax (33) -1-42 67 41 39

[1] E-mail address: otani@csinn.uucp
[2] E-mail address: simonin@csinn.uucp

1 Preamble

The Generation Module described here belongs to the SAGE system, which parses and generates sentences. SAGE is the Natural Language Frontend of the Dialogue Manager of the Esprit I project ESTEAM-316. ESTEAM-316 is an Advice-Giving system, and its testbed application field is private investment. Several examples given in this paper will be thus related to the financial domain.

One of the tasks of the generation process is to determine with which syntactic structure an idea will be expressed. Our synthesis module therefore generates a linguistic structure made of nested Syntactic Components (SCs) described by Functional Descriptions (FDs). An SC is characterised by its *meaning* slot and *is* composed of Syntactic sub-Components (sub-SCs). The value of the *meaning* slot is an instance of a semantic concept and is called a *token.* For instance, a clause SC is composed of the following sub-SCs: a subject, a verb, some complements and adverbials.

After an introduction to FDs in section 2, section 3 explains how linguistic knowledge has been structured using FDs. Section 4 describes how our generation module handles this knowledge whereas section 5 draws conclusions on the choices made in our implementation.

2 Introduction to Functional Descriptions

2.1 Functional Unification and Functional Descriptions

The theory of Functional Unification and Functional Descriptions (FDs) has been developed by Martin Kay [Kay 81] and is based on the formalism of Functional Grammar developed by Simon Dik [Dik 78]. Briefly, a FD is an object described by a set of s*lots,* i.e. an *attribute* to which a *value* is associated. The value of the slot is either atomic (integer, real, or string quoted between " "), or non-atomic (ordered list of objects quoted between ( ), non-ordered set of objects quoted between { }, nested FD or symbols). Figure 1 is an example of a FD.

```
[ age = 26
  height_in_meter = 1.83
  name = "Jimmy"
  chronology_of_professional_positions = ( software_engineer project_manager )
  hobbies = { skiing rock_climbing movies }
  present_position = [ function = project_manager
                       affiliation = Cap_Sogeti_Innovation ] ]
```

Figure 1: An example of FD.

Two other kinds of values which are *paths* and *links to other objects* have been introduced. The paths are either compiled when the object is loaded in memory, or dynamically computed when the loaded object is handled for instance by the generation process. They are quoted between < > [Fimbel & al 85]. The links are a kind of pointer to another object: they are invisible to the user and may be handled only by the interpreter (i.e. a program written in C).

2.2 Constraints: Conditions and Actions

We wanted our generation system to be declarative and reusable for other applications. For this, we needed to handle dynamic objects in different kinds of rules, especially in the generation grammar. These rules are described by FDs divided in three parts: a *condition* slot, a body of several slots that may be functionally unified with the current SC, and several slots of *conclusion.* Conditions and conclusions are lists of FDs containing *premise* and *action* slots. We may assess that there is an *or* logical operator between these FDs within the list. Condition, body and conclusions are not compulsory in a rule FD. However, no more than one condition is allowed, whereas there may be several conclusions. The operands of premises and actions are either constants (FD or any object known by the system) or dynamic objects specified by paths and links.

One of the most important links used by our system is a link pointing to the SC currently computed, which we shall symbolise by *current_SC.* Another one is a link from a sub-SC to its parent SC: a path like < *current_SC parent meaning* > will be interpreted as the meaning of the parent of the current SC. Examples of premises and actions will appear below.

A rule is relevant if one of the FDs of the *condition* slot is verified. If so, the body of the rule is unified to the current SC. Then all of the conclusions of the rule are evaluated: this means the actions of the first FD of a *conclusion* slot whose premises are verified are activated.

One might wonder about the usefulness of actions in a condition FD: this proved to be helpful in order to modify and prepare the current SC before it is unified with the body of the rule.

The following section details how the new FD formalism is used by SAGE to develop linguistic knowledge bases.

3 Linguistic knowledge bases

3.1 Lexicon-Grammar: syntactic valencies

The purpose of the lexicon-grammar is to define syntactic properties completely: using it, we are able to take into account a wide range of constructions of a given language. Our lexicon-grammar is based on the theory developed by Maurice Gross [Gross 75] and the studies carried out by the LADL on French constructions. To give an idea of this knowledge base, we give the information stored for the direct object of the verb *want* below.

- The standard construction is [Subject + Verb + Direct Object];
- The direct object may be a human being as in "The mother wants *her child",* a non-human entity as in "He wants *time",* or a *thai-clause* as in "Mary wants *that John settle down in Paris"*;
- The *thai-clause* is reduced to the following forms:
  - [Noun group + Adjective] or *NAdj,* if the omitted verb is *be:* e.g "The teacher wants *the exercise ready for tomorrow';*
  - [Verb in the complete infinitive form + complements] or *ToVinf0,* if the concept

of the subject of this clause is the same as that of the subject of *want:* e.g. "Mary wants *to settle down in Paris";*

• [Noun group + Verb in the complete infinitive form + complements] or *NToVinf* when the two subjects are different: e.g. "Mary wants *her friends to settle down in Paris";*

• The whole clause may be transformed into the passive form.

The semantic distributions are also defined in these FDs: they state to which semantic classes the token of a sub-SC may belong, according to *is_a* slots. For instance, in our lexicon-grammar, want allows a human being or a non-human item as the direct object.

In our formalism, this information is specified as shown in Figure 2.

```
Syntactic_want ↔
[ verb = [ word = want ]
  subject = [ noun_phrase_form = { ( noun_phrase + ) }
                 distribution = { ( *human + ) }
                 interrogative-pronoun = ( wh_who )]
  object1 = [ noun_phrase_form = { ( noun_phrase + )}
                 reduced_clause_form = { ( ToVinf0 + ) ( NAdj + )}
                 clause-form = { ( NToVinf + )}
                 distribution = { ( *human + ) ( *non_human + ) }
                 interrogative-pronoun = ( wh_what wh_who ) ]
  transformation = ( passive_transformation )
  ... ]
```

Figure 2: Characterization of *want* in the Lexicon-grammar.

The "+" symbol states that the corresponding form or semantic distribution is allowed. "-" would stand for forbidden constructions and "?" would introduce constructions that are acceptable for the parser, but dubious and forbidden in a generation processing.

The slot *interrogative_pronoun* defines the interrogative pronouns allowed for referring to the sub-SC in a Wh-questions.

Semantic distributions specified in the lexicon-grammar may be very detailed according to the use of such or such verb or noun. For instance, the subject of *to graze* may be a sheep, and that of *to eat* may be a human being. This information is very useful in synthesizing a pronoun: it helps to check whether a pronoun is ambiguous. If the speaker talks about a cow and a sheep, then an "it" before the verb *to graze* probably is the sheep, and an "it" before *to browse* refers to the cow: needless to say these distributions may greatly help the parser too.

3.2 Linguistic Definitions: a mapping from semantics slots to syntactic components

Given that identifiers of a token depend only on the concept of which it is an instance, the allocation of the slots in the sub-SCs raises the problem of the link between those specific identifiers and standard sub-SCs. This is solved with a *Linguistic Definition (LD),* which makes explicit the mapping between slots and Syntactic sub-Components (sub-SCs). A concept is thus associated with several LDs and the generator is able to choose among them according to such or such linguistic constraints.

For instance, the concept of **transaction* may be generated in a clause with the verb *to buy,* or with the verb *to sell.* In several cases, when the token of transaction is nested in another one, the noun phrase structure may be best adapted for instance as a subject of a sentence as in *"Her purchase of an expensive cottage* remained unknown for a long week".[1] The LD corresponding to the mapping from an instance of the concept **transaction* into the syntactic structure of the verb *sell* is described in Figure 3.

```
[ subject = [ meaning = < current_SC meaning seller > ]
  objectl =  [ meaning = < current_SC meaning object > ]
  object2 = [ meaning = < current_SC meaning buyer > ]
  gramma_-rule = gram_clause }
```

Figure 3: A Linguistic Definition.

Assuming that Figure 4 is the initial SC, and that the SC will be unified[2] with the LD of

```
[ meaning = [ instance_of = *transaction
               buyer = *user
               object = [ instance_of = *house
                          type = *cottage ] ]
```

Figure 4: An initial SC.

Figure 3, the path evaluation will result in the SC of Figure 5. After path evaluation, the sub-SCs with no *meaning* slot are deleted as in the case of the *object2* sub-SC in Figure 5. Concept names are marked with a star * for the sake of readability.

3.3 Specification of syntactic coding

The following two paragraphs give examples of how syntactic codes are specified using our "rule FD formalism".

3.3.1 Standard syntactic forms

The syntactic forms seen in §3.1 must be specified using a rule format. In our system, the *condition* slot of the symbol *ToVinf0* states that the construction [Verb in the complete

[1] We do not take into account pragmatic rules concerning focus, intention of the speaker, etc for choosing among several LDs.

[2] in the meaning of *functional unification.* See [Kay 81].

```
[ meaning = [ instance_of = *transaction
                seller = *user
                object = [ instance_of = *house
                              type = *cottage  ] ]
  subject = [ meaning = *user ]
  objectl = [ meaning = [ instance_of = *house
                              type = *cottage ] ]
  grammar_rule = gram_clause ]
```

Figure 5: An SC after an LD has been unified.

infinitive form + complements] is allowed if the subject of the current SC equals the subject of the main clause i.e. of the parent SC. In the body of *ToVinf0* (see Figure 6), the verb is set to the complete infinitive form, and the subject is erased because not expressed in an infinitive clause.

```
ToVinf0 →
[ condition = ( [ equal = ( < current_SC subject meaning >
                              < current_SC parent subject meaning > ) ] )
  subject = [ structure = erased ]
  verb = [ form = complete_infinitive ] ]
```

Figure 6: Description of the Syntactic form *ToVinf0.*

3.3.2 Interrogative pronouns

A slot *interrogative_pronoun* has been added in each sub-SC of the lexicon-grammar items, in order to generate the appropriate pronoun in a Wh-question. This is spectacular when the pronoun is not directly predictable from the syntactic function the sub-SC (subject, object or adverbials). For instance, starting from "The fund will be available *in 2 years.",* we may generate " *Under what delay* will the fund be available?".

In the case given in Figure 2, the *objectl* SC may be transformed using the code *wh_what* described by Figure 7.

```
wh_what →
[ condition = ( [ not_a = ( < current_SC meaning >
                              *human ) ] )
  pronoun = [ word = what ] ]
```

Figure 7: Description of the Syntactic code *wh_what.*

3.4 Other Knowledge Bases

Besides the KBs mentioned above, which are the Linguistic Definitions and the Lexicon-Grammar, there are mainly the generation grammar, the semantic dictionary and the lexicon of words. The last two will not be described here:

- The semantic dictionary is a semantic net of FDs linked with *is_a* slots. Briefly, the concepts are defined with FDs divided in three parts: 1) the semantic net link descriptions; 2) a *schemata* which lists the slots that characterize a token, i.e. an instance of the concept; and 3) the list of LDs that the generation module has at its disposal for the given token.
- The lexicon is a standard dictionary of English words with the indication of syntactic categories (noun, preposition, auxiliary, verb, etc) and of conjugation models (the verb *to eat* obeys the same conjugation rules as *to sing*).

The following section details the grammar structure and the generation process itself.

4 Generation strategies

4.1 Generation grammar rules

For a given generation rule, the grammar specifies under what conditions it may be applied using the *slot condition*, what rules are to be chosen for the synthesis of each sub-SC in the FD of the sub-SC of the body of the rule, and what actions are to be carried out on the current SC (such as choosing the number and person of a verb according to the subject within a clause).

Figure 8 is an example of what may be specified for the clause grammar rule.

In our implementation, neither *condition* nor *conclusion* slots are needed in the clause rule. When provided, they indicate in which cases a grammar rule is relevant to the current SC, and if so, what actions are to be undertaken on the SC.

The names of the sub-SCs are made explicit using the slot *list_of_subSCs.* Sub-SCs are not all compulsory and may be deleted in the SC if no *meaning* is given to them after the unification of the LD (see §3.2). The parent of the sub-SCs is the current SC: it is specified so by the path < *current_SC* > in the slot *parent.*

Figure 8 shows also how the slot *grammar_rules* specifies the grammar rules allowed for each sub-SC. One sub-SC may be synthesized simply using another grammar rule. For instance, objects of a clause may be synthesized as personal pronoun using *gram_pers_pron* rule or as reflexive pronoun using *gram_reflexive* rule. Most of the time, the sub-SCs are generated using LD and lexicon-grammar as stated by the symbol *linguistic_synthesis.* The generation rules are tried in the order specified by *grammar_rules.*

The *order* slots specify the order of the sub-SCs in the generated sentence, and may be modified by actions. For instance, if the *object2* sub-SC is synthesized as a personal pronoun, then it is put before *objectl* by decreasing its *order* value form 210 to 110. The *order* values are interpreted by the morphological generator i.e. all sub-SCs of an SC are sorted according to their order, before the corresponding words are generated.

```
gram_clause ↔
[ list_of_subSCs = ( subject verb objectl object2 adverbial_date )
  rule_type = clause_type
  subject = [ parent = < current_SC >
              order = 50
              grammar_rules = ( gram_pers_pron linguistic_synthesis ) ]
  verb = [ parent = < current_SC >
           order = 60
           grammar_rules = ( gram_verbe ) ]
  objectl = [ order = 200
              parent = < current_SC >
              grammar_rules = ( gram_reflexive gram_pers_pron linguistic_synthesis ) }
  object2 = [ order = 210
               parent — < current_SC >
              grammar_rules = ( gram_reflexive gram_pers_pron linguistic_synthesis ) ]
  adverbial_date = [ parent = < current_SC >
                      order =400
                     grammar_rules = ( gram_pers_pron linguistic_synthesis ) ] ]
```

Figure 8: An example of a clause grammar rule.

4.2 Interpretation in the generation process

The generation process is top-down, with backtracking. It recursively builds a complex object of several nested SCs. At the beginning of the generation process, the first SC is an object similar to:
[*meaning* = *token_to_be_generated*]

The current SC is supplemented with sub-SCs in a loop: according to the concept of the token in the meaning slot, the generation interpreter chooses a LD, then a syntactic structure in the lexicon-grammar. These two FDs are functionally unified with the current SC.

Then, one syntactic form like *noun_phrase* or *ToVinf0* (see §3.3.1) is chosen for the current SC according to the grammar rule given in the LD and the validity condition of the syntactic form. These forma are compulsory only if the type of the current SC is of clause or noun phrase. In the former case, the generator needs either a complete clause form or a reduced one and chooses among the lists of *reduction_form* or *clause_form;* in the second case, it requires a *noun_phrase_form.* If found, the form is functionally unified with the current SC. Figure 9 shows the SC corresponding to *You want time* after functional unifications. The concept **user* is generated in a dialog pronoun into the second person (*you*) because the Natural Language Front-End is integrated with a Person-Machine Dialogue application.

At this stage of the process, the generator may add several modifiers to the current level, adverbials in clauses, or adjectives in noun groups: these adjuncts are also carried through functional unifications since the modifiers are also described in a FD just like any LD. Henceforth, the modifiers are handled in the same way as the sub-SCs defined by the LD.

Lastly, the grammar rule specified by the LD is applied. First, the body of the grammar rule is functionally unified to the current SC. Secondly, the sub-SCs are synthesized either by another grammar rule such as the one allowing the pronominalization of sub-SCs, or by the

```
[ meaning = [ instance_of = *want
                   actor = *user
                   object = [ instance_of = *time ] ]
  subject = [ meaning = *user
                 distribution = {(*human +)}
                noun_phrase = {(noun_phrase +)} ]
  verb = [ word = want ]
  objectl = [  meaning = [ instance_of = *time ]
                  distribution = {(*non_human +) (*human +)}
                  reduction-form = {(ToVinf0 +) (NAdj +)}
                  clause_form = {(NToVinf +)}
                  noun_phrase_form = {(noun_phrase +)}             ]
  transformation = {(passivel 100)}                                ]
```

Figure 9: Syntactic Component of *You want time.*

generation process described in the present paragraph §4.2, according to the *grammar_rules* list. It is during that phase that the sub-SCs become the *current SC* in turn.

This is where our declarative KBs based on Functional Descriptions prove to be efficient. The same heuristic based on series of functional unifications is used for totally different structures such as noun phrase or clause. Therefore, this loop is allowed to be totally recursive.

Then the conclusion slots of the grammar rule are evaluated, if there are any.

Transformations are processed whenever they are needed, as for questions (which puts the verb in the interrogative form and inserts an auxiliary verb before the subject), or negations or passive transformations.

If a failure occurs during this loop, for instance if one of the sub-SC has not been generated completely, then backtracking is carried out by choosing another LD and/or another grammar rule.

4.3 Conclusion on parsing and generation grammars

The main feature of our Natural Language Front-End is that the Parsing and Generation processes are carried out using the same linguistic knowledge bases. On the other hand, parsing and generation grammar formalism differ because of their dedicated heuristics. Unlike parsing, our generation process is not a sequence of "left-to-right" procedures [Danlos 87a][Danlos 87b]. Moreover, a given heuristic of clause transformation is strongly dedicated to a parsing or to a generation process: during generation, the order of the objects in a clause may be modified after they have been synthesized into personal pronouns (see §4.2); during parsing, it is not possible to recognize an *objectl* and an *object2* unless the aphorisms are solved.

5 Conclusion on the implementation

5.1 Declaration and interpretation

Rules may be declaratively defined using *condition* and *conclusion* slots. This has been made possible with the link *current_SC* and with the slot *parent* referring to the embedding SC, allowing dynamic paths like < *current_SC parent meaning* >.

Yet some of the generation inferences are invisible to the user for being integrated in the C programs. For instance, the choice of a syntactic form (*ToVinf0, NAdj,* etc) is not declared in a grammar rule (clause or noun phrase rule) but is processed by a specific C function called before the grammar rule is activated. This implicit inference might become visible and might be declared in the generation grammar if an action *choose_a_form_among* were introduced, with the following format:

choose_a_form_among = < *current_SC noun_phrase_form* >. Other premises or actions may be added to the inference engine in the same way.

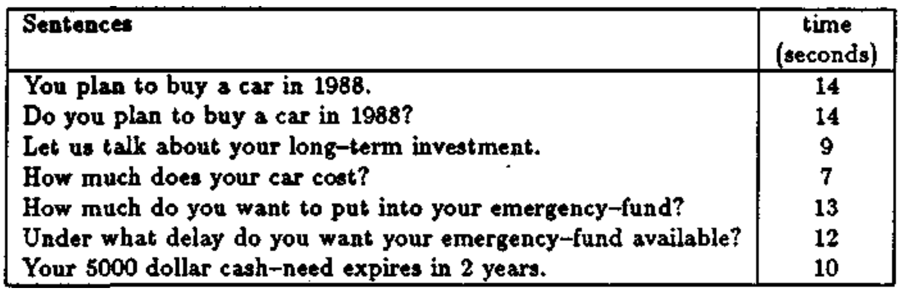
5.2 Backtracking

The concept of link in FD is useful for handling objects defined dynamically with paths, and also for backtracking.

If a grammar rule fails for instance, then we must be able to recover a former state of the current SC i.e. to destroy slots that have been added by functional unification. So each SC is actually handled as a link to a FD. Before any inference is carried out, the FD of each SC is copied and stored in a backup FD. In case of failure of the generation process, the backtracking process makes the SC links point to the latter stored FDs: the same links (i.e. SC pointers) are thus still relevant.

5.3 Results and performance

Presently, we are able to generate sentences with infinitive clauses, imperative clauses, Yes-No questions, several Wh-questions (where the interrogative pronoun is bound either to the main clause or to a nested clause). Here is a table of several generated sentences together with an average response time they require on a SUN 3/50.



Sentences	time (seconds)
You plan to buy a car in 1988.	14
Do you plan to buy a car in 1988?	14
Let us talk about your long–term investment.	9
How much does your car cost?	7
How much do you want to put into your emergency–fund?	13
Under what delay do you want your emergency–fund available?	12
Your 5000 dollar cash–need expires in 2 years.	10

We believe that we brought the theory of Functional Description and Functional Unification some promising features, especially concerning link handling and rule description. The constitution of declarative knowledge using dynamic operands and declarative rules is a principle that may be applied in other fields of Natural Language Processing. This allow the sharing of linguistic knowledge bases by parser and generation module, even though there are distinct parsing and generation grammars, and dedicated interpreters in C.

Moreover, we believe that FDs as described in our present paper may be useful for realms of Knowledge based systems other than Natural Language Processing.

References

[Danlos 87a] Laurence Danlos, *A French and English Syntactic Component for Generation,* Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics, Kempen G. ed, Dordrecht/Boston, Martinus Nijhoff Publishers, 1987.

[Danlos 87b] Laurence Danlos, *The linguistic basis of text generation,* Cambridge University Press.

[Dik 78] Simon Dik, *Functional Grammar,* Publications in Language Science, Foris Publications, Dordrecht, Holland, 1978.

[Fimbel & al 85] Eric Fimbel, Herbert Grosco, Jean-Marie Lancel, Nathalie Simonin, *Using a Text Model for Analysis and Generation,* Proceedings of ACL'85, Geneva, pp 226-231.

[Gross 75] Maurice Gross, *Méthodes en syntaxe, Régime des constructions complétives,* Hermann, 1975.

[Kay 81] Martin Kay, *Unification Grammars,* Xerox Publication, 1981.

[Lancel & al 86] Jean-Marie Lancel, Francois Rousselot, Nathalie Simonin, *A Grammar used for Parsing and Generation,* 11th International Conference on Computational Linguistics, Proceedings, Coling'86, August 1986.

[Lancel & al 88] Jean-Marie Lancel, Miyo Otani, Nathalie Simonin, Laurence Danlos *SAGE: a Sentence Parsing and Generation System,* "Sentence Parsing and Generation with a Semantic Dictionary and a Lexicon-Grammar", 13th International Conference on Computational Linguistics, Proceedings, Coling'88, August 1988.

[Longman 78] *Longman Dictionary of Contemporary English,* Longman Group Limited, 1978, Corrections 1981.

[Otani 88] Miyo Otani, Jean-Marie Lancel, *Sentence Generation: from semantic representations to sentences throughout Linguistic Definitions and Lexicon-Grammar,* Proceedings of Ecai '88, August 1988.