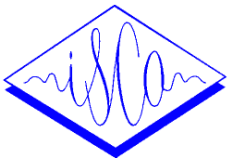

6th International Conference on Spoken Language Processing (ICSLP 2000)
Beijing, China
October 16-20, 2000

FORMAL AND NATURAL LANGUAGE GENERATION IN THE MERCURY CONVERSATIONAL SYSTEM[1]

Stephanie Seneff and Joseph Polifroni

Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139 USA

ABSTRACT

This paper describes the generation component of our MERCURY flight reservation conversational system. Generation makes use of the GENESIS-II generation server, which represents a significant redesign from its predecessor, GENESIS. While the main focus is on response generation, we also discuss a variety of other generation needs that are fulfilled by GENESIS-II. These include a paraphrase of the user query back into English and into a flattened electronic form, the paraphrase of the electronic form into a database query language, and the conversion of flight tables into HTML format.

1. INTRODUCTION

Over the past two years, we have been developing a conversational system that provides access to real flight information, world wide. The system, called Mercury, which has been implemented within the GALAXY communicator architecture [4], adopts a mixed-initiative dialogue model in which the system prompts for information but the user is not required to comply.

In [5], we described the Mercury dialogue manager, which makes use of a dialogue control table to plan each turn's system response. This paper focuses on language generation, which plays a critical role in many aspects of system processing.

Language generation in all of the conversational systems we have developed thus far has made use of the GENESIS language generation system developed in our group [2]. Over the past year, we have implemented a new version of GENESIS, called GENESIS-II, based on our experience in using GENESIS for multiple natural and formal languages. GENESIS-II has eliminated many of the idiosyncracies of GENESIS and thus provides a more intuitive user interface. It has several additional features that make it considerably more powerful. Like GENESIS, GENESIS-II makes use of three distinct control files for each unique generation language: a grammar file, a lexicon, and a rewrite-rules file. We will elaborate later in this paper on some of the particulars of the control files. A more detailed description of the GENESIS-II formalism can be found in a companion paper [1].

All generation activities are invoked through a small set of operations defined for the generation server, and mediated via the control program in the GALAXY hub. Typically, a rule in the hub program specifies an input frame for generation, a domain, and a generation language, and the GENESIS-II server consults the appropriate tables to produce a generation string, which it returns to the hub for subsequent rerouting.

Role	Input	Output
Response Generation	Response Frame Response Frame	English Text Synthesized Speech
Paraphrase Generation	Input Frame Input Topic	English Text Synthesized Speech
Formal Language Generation	Input Frame Electronic Form Flight Table	Electronic Form DB Query HTML Table

Figure 1: Various generation roles in Mercury that are handled by the GENESIS-II server.

There are several distinct roles that GENESIS-II plays in the MERCURY system, as indicated in Table 1. One of the most important roles is to generate a natural language response for the user. GENESIS-II uses a common grammar file but distinct lexicons to convert the response frame into a well-formed English text to display in the graphical interface and into a marked-up text format for further processing by the ENVOICE speech synthesizer [3], to produce the response speech waveform.

Often, just before going to the database, MERCURY paraphrases back to the user the topic of their question, as in, "Okay, United flights from Boston to Dallas on March third. One moment please." In displayful mode, the full paraphrase of the user query is displayed in a special paraphrase window. These mechanisms serve to inform the user the degree to which MERCURY understood the question. The full paraphrase is especially difficult for wh-queries, where a trace mechanism is necessary to move the wh-marked noun phrase to the front of the surface form string.

GENESIS-II also handles several instances of paraphrases into formal languages. The first is to convert the linguistic frame (in context) that the NL component produces into a flattened *electronic form* (henceforth, *e*-form), a representation that is more convenient for the dialogue manager to interpret. In this case, GENESIS-II produces a string in a simple mark-up language, which is then converted into an *e*-form frame by a GALAXY library routine. Another formal language task is to convert the *e*-form into the database query appropriate for database retrieval. Finally, in displayful mode, it converts the retrieved list of flights into a clickable HTML table for graphical display.

The remainder of the paper is organized as follows. We will be-

[1]This work was supported by DARPA under contract N66001-99-1-8904 monitored through Naval Command, Control and Ocean Surveillance Center.

(a)	speak_multiple_first_departure	:comments !three_plus :topic . >first_departure >and_arrives_at
(b)	first_departure	!first_departure :departure_time
(c)	and_arrives_at	!and_arrives_at ($Low :arrival_time)
(d)	topic_template	>prepreds $core >postpreds

Figure 2: Selected grammar rules for the example of Figure 4.

gin with a description of the response generation mechanisms in MERCURY, as this is the most obvious generation task for conversational systems. We will follow this with a discussion on user query paraphrase generation. Finally, we will discuss the formal language generation roles of GENESIS-II. Throughout, we will highlight through selected examples how MERCURY deals with some of the difficult cases, such as multiple word senses, lists, conjunction, prosodic marking, movement, etc.

(a)	no_nonstops	"There are no nonstop flights."
(b)	three_plus	"There are more than three"
(c)	first_departure	"The first flight leaves at"
(d)	and_arrives_at	"and arrives at"
(e)	DFW	"Dallas Fort Worth"
(f)	UA	"United"

Figure 3: Selected lexical entries for the example of Figure 4.

2. Response Generation

Response generation is a crucial aspect of conversational systems, particularly displayless systems which depend fully on the spoken response for communication. In MERCURY, response generation is carried out in two distinct stages, where the first stage is handled by the dialogue management server, and the second *surface generation* stage is handled by GENESIS-II.

The dialogue manager produces a response *frame*, which is a representation of the meaning of the intended response. It is the responsibility of GENESIS-II to convert response frames into both well-formed English strings for display in a graphical window, and a marked-up string for interpretation by the ENVOICE speech synthesizer [3]. For this purpose, GENESIS-II makes use of a grammar file and a pair of lexicon files, one for each mode of generation. The grammar file provides recursive grammar rules that dictate the ordering of constituents, and the lexicon converts named entities into surface form strings.

An example frame is given in Figure 4, along with selected entries from the grammar and lexicon files in Figures 2 and 3 respectively. There are three main linguistic categories for frames: *topic*, *predicate*, and *clause*, each of which has a default template for its generation rule, which is invoked if its name is missing from the grammar rules. Generation begins with the top level clause constituent, which looks up its generation message based on its name, and constructs a string by concatenating subconstituents in the order indicated by the grammar rule (see Figure 2(a)). If it finds no entry, it backs off to the generic template. A representative default topic template is included in Figure 2(d). The rules for "prepreds" and "postpreds" (not shown in the figure) enumerate all the possible predicates that precede and follow the main noun of a topic (denoted by $core) respectively.

```
{c speak_multiple_first_departure
   :topic {q flight
             :number "pl"
             :pred (
             {p source
                 :topic {q city
                           :name "SGF" } }
             {p destination ... }
             {p airline ... }
             {p month_date
                  :topic {q date
                            :month "JUL" ... } }

             {p depart_between
                  :topic {q time
                             :hour 12
                             :xm "pm"
                             :and {q time
                                      :hour 6
                                      :xm "pm" } } }
             )
          }
   :departure_time "1:35 p.m."
   :arrival_time "5:01 p.m."
   :comments ( {c no_nonstops } ) }
```

Figure 4: Example response frame for the MERCURY system. This frame generates as: "There are no nonstop flights. There are more than three United flights from Springfield to Dallas Fort Worth departing between noon and 6:00 pm on Monday July 17. The first flight leaves at 1:35 p.m. and arrives at 5:01 p.m."

The notation ">" is a "goto" command, that recursively fills in substrings in the main generation rule. The "!" notation directs GENESIS-II to look up the subsequent string directly in the lexicon (refer to Figure 3). Keys to be accessed in the input frame are denoted by a ":". The contents of these keys are recursively evaluated through the rules to fully expand the frame into a string.

The *selector*, $Low in the "and_arrives_at" rule (Figure 2(c)) is a mechanism to select for a low prosodic tone in the arrival time generation, because this string occurs at the end of the sentence. A similar mechanism is also used to disambiguate multiple senses for a lexical entry. For instance, "AUG," which defaults to "August," is looked up under $Cty as "Augusta, Maine." Another example is the "$ord" selector, used to select "seventh" instead of "seven" for a date frame.

The response frame typically contains a number of unique elements, most of which are optional. The name of the frame identifies the top level clause of the message. Its contents may include a list of one or more comments, accumulated in a *:comments* list structure via a GALAXY library routine (an example is the "no_nonstops" message in the illustrated response frame). It often also contains a *:continuant* message, which by default produces "something_else" (e.g., "Is there something else I can do for you?"), but which can be set to a number of other pos-

flight_list	>first >butlast >last
first	$first :first ,
butlast	:butlast ,
last	$last !and :last

Figure 5: Sample entries from the grammar file for generation of spoken responses to read off a list of items.

sibilities, such as "Please choose one," "Would that work?" or "Shall I price your itinerary?"[2] There is also typically a *:topic* entry, representing a high-level description of the set of flights being offered. The frame under the topic could be simply a copy of the linguistic frame present in the user query, or it could also be generated by the dialogue manager, as a generic description of the set of extracted flights. For example, if all the flights connect in the same city, then the topic would contain information about the connection city, which would then only be mentioned once. Finally, the response might contain either a list of *flights* to be suggested as options to the user or a set of *attributes* to be spoken about for a particular flight that is singled out. The dialogue manager decides what flight or flights to mention based on constraints such as how many flights are available altogether, how many of them are nonstop, etc.

GENESIS-II provides a very convenient framework for handling lists, as illustrated in Figure 5. The first and last items in the list are specially marked as ":first" and ":last" respectively. Other entries are referred to as ":butlast". The $first and $last selectors scope over every item in the rule and all descendents, and are used by the synthesizer for prosodic selection.

The generation vocabulary file for the synthesizer includes some entries whose surface form is a direct pointer into a pre-recorded waveform. In other cases, the synthesizer is able to select appropriate generation units automatically by searching through a Finite State Transducer network, as described in [3]. As alluded to earlier, there are often multiple entries for the same word, indexed under differing prosodic selectors.

3. English Paraphrases

The query paraphrase, only invoked in displayful mode, is actually one of the more difficult generation tasks, due to the fact that, in English, a wh-marked constituent is moved in the surface form realization to the front of the sentence. In linguistic terminology, the moved constituent leaves behind a "trace," as in, "<what time> will the United flight arrive <trace>?"

In database query domains, wh-marked questions are frequently encountered. In English, an appropriate linguistic analysis repositions this constituent in its deep structure location, thus establishing, in the example, that "what time" refers to an arrival time. GENESIS-II must recreate the surface form generation by inserting the paraphrase for the string "what time" at the beginning of the generated string. This is accomplished by making use of a specially marked generation tag (preposed with "−−"), which instructs a particular constituent to *construct* but not *place* a generation string. Instead, the string is made accessible to higher level constituents, which can then control its placement order.

[2] Although messages are described here as fully formed English strings for expository purposes, they are actually named clauses which paraphrase to the appropriate string via a lexical entry in GENESIS-II.

Grammar Rules

wh_query	−−trace :aux :topic >main_preds
main_preds	. . . arrival_time . . .
arrival_time	$core :topic
topic_template	:quant ($if :trace > −−trace >np)
−−trace	:trace $core
np	:quant >prepred $core >postpred

Lexicon

which	Q	. . . :trace "what" . . .
do	X	. . . THIRD "does" :MODE "root" . . .
arrival_time	V	. . . ROOT "arrive". . .

Figure 6: Example entries from the grammar and lexicon files to generate the paraphrase string for the frame in Figure 7.

Linguistic Frame:

```
{c wh_query :mode "finite" :num "sing" :aux "do"
   :topic
      {q flight
         :quant "def"
         :pred {p airline
                  :topic {q airline_name
                            :name "united" } } }
   :pred {p arrival_time
            :topic {q time :quant "which" } } }
```

Corresponding Electronic Form:

airline: UA arrival_time: which

Figure 7: Example linguistic frame representation for the query "What time does the United flight arrive?" and the corresponding *e*-form.

Figure 6 gives entries from the grammar and lexicon files needed to paraphrase the example linguistic frame in Figure 7. The top-level clause is named *wh_query*, and it will ultimately generate a sequence of four constituents as follows: "what time" (−−trace) "does" (:aux) "the United flight" (:topic), and "arrive" (>main_preds). The "−−trace" constituent was actually pre-generated by the "{q time :quant "which"} frame, under the ":topic" key in the *arrival_time* predicate, but handed up to the *wh_query* frame for placement. This is accomplished by consulting the *topic_template* entry in the template file. First, the ":trace" key is inserted into the frame upon generation of the "which" vocabulary entry for the quantifier. The :topic template specifies, using the "$if" construction, that the *−−trace* template should be invoked if a ":trace" key exists in the frame. Otherwise a normal *np* template should be generated and immediately placed. Because of the presence of the ":trace" key, the *time* topic generates nothing in place, but preconstructs the "what time" phrase for the *wh_query* template to insert.

Another notable aspect of the generation process is that the auxiliary "do" sets the mode for the verb "arrival_time" to *root*, which controls the selection of the appropriate inflectional ending.

Grammar Rules

statement	:topic >main_preds >and_clause
and_clause	"and: <start>" :and "<end>"

Resulting *E*-form

```
{c eform  :source "BOS" :destination "DFW"
   :day "tomorrow" :depart_interval  "morning"
   :and {c eform
           :return_date "MAY03"
           :depart_interval "evening" } }
```

Figure 8: Selected rules from the template file for *e*-form generation and resulting *e*-form for the query "I want to go from Boston to Dallas tomorrow morning and return on May third in the evening."

4. Formal Language Generation

GENESIS-II is responsible for generating not only linguistic strings but also several outputs that are expressed in formal languages, as indicated in Figure 1. GENESIS-II is responsible for converting the user's query frame into a flattened elctronic form, and subsequently, converting the electronic form into a database query. Finally, the tabular results retrieved from the database are displayed as an HTML table in the gui interface, augmented with clickable icons for selected airlines and airports.

Electronic Form The linguistic frame is not a particularly convenient representation for the dialogue manager to use for further analysis. Instead, a much simpler *e*-form representation is derived using GENESIS-II, representing the constraints as a set of keys and associated values. This is a very easy generation task, as nearly all predicates can make use of a generic template according to the following grammar rule:

predicate_template $core ":" :topic[3]

Figure 7 shows, along with the *linguistic* frame, the corresponding *e*-form generation string. for the user query, "What time does the United flight arrive?"

The *e*-form is not always completely free from hierarchy. To handle clause level conjunction, we created a very simple markup language, surrounding a generation string with "<start>", "<end>" to indicate an internal frame. Thus, the sentence, "I want to go from Boston to Dallas tomorrow morning and return on May third in the evening" produces a generation string containing the entry: 'and: <start> :return_date: "MAY03" :depart_interval "morning" <end>,' resulting in the *e*-form shown in Figure 8, which isolates the return date from the remainder of the attributes.

Database Query Database query generation is probably the most straightforward task for GENESIS-II in MERCURY. The input is typically an *e*-form, and the output is a string of field entries separated by commas, arranged in a prescribed order, according to the format dictated by our database access script. The user enrollment data are stored in an SQL table, whose generation requires are also very straightforward.

HTML Tables The database returns the database entries as a list of frames represented as *e*-forms. In displayful mode, the full set of flights is sent to GENESIS-II to be converted into an HTML format for display. The system presents list items that can be mouse-clicked and referred to anaphorically, as well as individual table items that link to appropriate Web pages for airlines and airports. For example, the HTML grammar rule for *airline* might appear as follows:

airline "<TD align=center>" :airline , :flight_number

and the corresponding lexical entry might be:

AA "<a href=http://www.aa.com>
<img src= "aa.gif" < /img>< /a>"

5. Summary

This paper has focused on the generation component of our MERCURY system, GENESIS-II. We believe it is a unique aspect of GENESIS-II that it is called upon to generate not only natural language strings but also formal language strings and marked-up strings for speech synthesis. The design of GENESIS-II has been guided by many years of experience using its predecessor, GENESIS. We feel that GENESIS-II provides mechanisms that deal with linguistic and non-linguistic inputs in a common, integrated, framework, naturally supporting high quality generation from structures that intermix both representations. We have found that GENESIS-II was able to effectively handle all the generation needs of MERCURY.

6. ACKNOWLEDGMENTS

We are indebted to Lauren Baptist, who developed and implemented the GENESIS-II system, and who was always very responsive to our needs for enhanced system capabilities. Philipp Schmid implemented the original HTML generation in GENESIS which was easily adapted to GENESIS-II.

7. REFERENCES

1. L. Baptist and S. Seneff, "GENESIS-II: A Versatile System for Language Generation in Conversational System Applications," *These Proceedings*, Beijing, China, 2000.
2. J. Glass, J. Polifroni and S. Seneff, "Multilingual Language Generation Across Multiple Domains," *Proc. ICSLP '94*, pp. 983–986, Yokohama, Japan, Sept. 1994.
3. J. Yi, J. Glass, and I. L. Hetherington, "A Flexible, Scalable Finite-State Transducer Architecture for Corpus-Based Concatenative Speech Synthesis," *These Proceedings.*, Beijing, China, Oct. 2000.
4. S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "Galaxy-II: A Reference Architecture for Conversational System Development," *ICSLP '98*, pp. 931-934, Sydney, Australia, December, 1998.
5. S. Seneff and J. Polifroni, "Dialogue Management in the Mercury Flight Reservation System," Proc. *ANLP-NAACL 2000*, Seattle, WA, May, 2000.

[3]The extra space between the predicate's name and the colon is removed through a rewrite rule.