A Rule-based Conversation Participant

Robert E. Frederking
Computer Science Department, Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The problem of modeling human understanding and generation of a coherent dialog is investigated by simulating a conversation participant. The rule-based system currently under development attempts to capture the intuitive concept of "topic" using data structures consisting of declarative representations of the subjects under discussion linked to the utterances and rules that generated them. Scripts, goal trees, and a semantic network are brought to bear by general, domain-independent conversational rules to understand and generate coherent topic transitions and specific output utterances.

1. Rules, topics, and utterances

Numerous systems have been proposed to model human use of language in conversation (speech acts [1], MICS [3], Grosz [5]). They have attacked the problem from several different directions. Often an attempt has been made to develop some intersentential analog of syntax, despite the severe problems that grammar-oriented parsers have experienced. The program described in this paper avoids the use of such a grammar, using instead a model of the conversation's topics to provide the necessary connections between utterances. It is similar to the ELI parsing system, developed by Riesbeck and Schank [7], in that it uses relatively small, independent segments of code (or "rules") to decide how to respond to each utterance, given the context of the utterances that have already occurred. The program currently operates in the role of a graduate student discussing qualifier exams, although the rules and control structures are independent of the domain, and do not assume any *a priori* topic of discussion.

The main goals of this project are:

- To develop a small number of general rules that manipulate internal models of topics in order to produce a coherent conversation.
- To develop a representation for these models of topics which will enable the rules to generate responses, control the flow of conversation, and maintain a history of the system's actions during the current conversation.
- To integrate information from a semantic network, scripts, dynamic goal trees, and the current conversation in order to allow intelligent action by the rules.

The rule-based approach was chosen because it appears to work in a better and more natural way than syntactic pattern matching in the domain of single utterances, even though a grammatical structure can be clearly demonstrated there. If it is awkward to use a grammar for single-sentence analysis, why expect it to work in the larger domain of human discourse, where there is no obviously demonstrable "syntactic" structure? In place of grammar productions, rules are used which can initiate and close topics, and form utterances based on the input, current topics, and long-term knowledge. This set of rules does not include any domain-specific inferences; instead, these are placed into the semantic network when the situations in which they apply are discussed.

It is important to realize that a "topic" in the sense used in this paper is not the same thing as the concept of "focus" used in the anaphora and coreference disambiguation literature. There, the idea is to decide which part of a sentence is being focused on (the "topic" of the sentence), so that the system can determine which phrase will be referred to by any future anaphoric references (such as pronouns). In this paper, a topic is a concept, possibly encompassing more than the sentence itself, which is "brought to mind" when a person hears an utterance (the "topic" of a conversation). It is used to decide which utterances can be generated in response to the input utterance, something that the focus of a sentence (by itself) can not in general do. The topics need to be stored (as opposed to possibly generating them when needed) simply because a topic raised by an input utterance might not be addressed until a more interesting topic has been discussed.

The data structure used to represent a topic is simply an object whose value is a Conceptual Dependency (or CD) [8] description of the topic, with pointers to rules, utterances, and other topics which are causally or temporally related to it, plus an indication of what conversational goal of the program this topic is intended to fulfill. The types of relations represented include: the rule (and any utterances involved) that resulted in the generation of the topic, any utterances generated from the topic, the topics generated before and after this one (if any), and the rule (and utterances) that resulted in the closing of this topic (if it has been closed). Utterances have a similar representation: a CD expression with pointers to the rules, topics, and other utterances to which they are related. This interconnected set of CD expressions is referred to as the topic-utterance graph, a small example of which (without CDs) is illustrated in Figure 1-1. The various pointers allow the program to remember what it has or has not done, and why. Some are used by rules that have already been implemented, while others are provided for rules not yet built (the current rules are described in sections 2.2 and 3).

This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

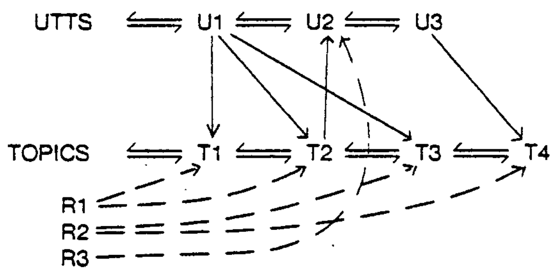

Figure 1-1: A topic-utterance graph

2. The computational model

The system under implementation is, as the title says, a rule-based conversation participant. Since language was originally only spoken, and used primarily as an immediate communication device, it is not unreasonable to assume that the mental machinery we wish to model is designed primarily for use in an interactive fashion, such as in dialogue. Thus, it is more natural to model one interacting participant than to try to model an external observer's understanding of the whole interaction.

2.1. Control

One of the nice properties of rule-based systems is that they tend to have simple control structures. In the conversation participant, the rule application routine is simply an initialization followed by a loop in which a CD expression is input, rules are tried until one produces a reply-wait signal, and the output CD is printed. A special token is output to indicate that the conversation is over, causing an exit from the loop. One can view this part of the model as an input/output interface, connecting the data structures that the rules access with the outside world.

Control decisions outside of the rules themselves are handled by the agenda structure and the interest-rating routine. An agenda is essentially a list of lists, with each of the sublists referred to as a "bucket". Each bucket holds the names of one or more rules. The actual firing of rules is not as simple as indicated in the above paragraph, in that all of the rules in a bucket are tested, and allowed to fire if their test clauses are true. After all the rules in a bucket have been tested, if any of them have produced a reply-wait signal, the "best" utterance is chosen for output by the interest-rating routine, and the main loop described above continues. If none have indicated a need to wait, the next bucket is then tried. Thus, the rules in the first bucket are always tried and have highest priority. Priority decreases on a bucket-by-bucket basis down to the last bucket. In a normal agenda, the act of firing is the same as what I am calling the reply-wait signal, but in this system there is an additional twist. It is necessary to have a way to produce two sentences in a row, not necessarily tightly related to each other (such as an interjection followed by a question). Rather than trying to guarantee that all such sets of rules are in single buckets, the rules have been given the ability to fire, produce an utterance, cause it to be output immediately, and not have the agenda stopped, simply by indicating that a reply-wait is not needed. It is also possible for a rule to fire without producing either an utterance or a reply-wait, as is the case for rules that simply create topics, or to produce a list of utterances, which the interest-rater must then look through.

The interest-rating routine determines which of the utterances produced by the rules in a bucket (and not immediately output) is the best, and so should be output. This is done by comparing the proposed utterance to our model of the goals of the speaker, the listener, and the person being discussed. Currently only the goals of the person being discussed are examined, but this will be extended to include the goals of the other two. The comparison involves looking through our model of his goal tree, giving an utterance a higher ranking for matching a more important goal. This is adjusted by a small amount to favor utterances which imply reaching a goal and to disfavor those which imply failing to reach it. Goal trees are stored in long-term memory (see next section).

2.2. Memories

There are three main kinds of memory in this model: working memory, long-term memory, and rule memory. The data structures representing working memory include several global variables plus the topic-utterance graph. The topic-utterance graph has the general form of two doubly-linked lists, one consisting of all utterances input and output (in chronological order) and the other containing the topics (in the order they were generated), with various pointers indicating the relationships between individual topics and utterances. These were detailed in section 1.

Long-term memory is represented as a semantic network [2]. Input utterances which are accepted as true, as well as their immediate inferences, are stored here. The typical semantic network concept has been extended somewhat to include two types of information not usually found there: goal trees and scripts.

Goal trees [6, 3] are stored under individual tokens or classes (on the property GOALS) by name. They consist of several CD concepts linked together by SUBGOAL/SUPERGOAL links, with the top SUPERGOAL being the most important goal, and with importance decreasing with distance below the top of the goal tree. Goal trees represent the program's model of a person or organization's goals. Unlike an earlier conversation program [3], in this system they can be changed during the course of a conversation as the program gathers new information about the entities it already knows something about. For example, if the program knows that graduate students want to pass a particular test, and that Frank is a graduate student, and it hears that Frank passed the test, it will create an individual goal tree for Frank, and remove the goal of passing that test. This is done by the routine which stores CDs in the semantic network, whenever a goal is mentioned as the second clause of an inference rule that is being stored. If the rule is stored as true, the first clause of the implication is made a subgoal of the mentioned goal in the actor's goal tree. If the rule is negated, any subgoal matching the first clause is removed from the goal tree.

As for scripts [9], these are the model's episodic memory and are stored as tokens in the semantic network, under the class SCRIPT. Each one represents a detailed knowledge of some sequence of events (and states), and can contain instances of other scripts as events. The individual events are represented in CD, and are generally descriptions of steps in a commonly occuring routine, such as going to a restaurant or taking a train trip. In the current context, the main script deals with the various aspects of a graduate student taking a qualifier. There are parameters to a script, called "roles" - in this case, the student, the writers of the exam, the graders, etc. Each role has some required preconditions. For example, any writer must be a professor at this university. There are also postconditions, such as the fact that if the student passes the qual he/she has fulfilled that requirement for the Ph.D. and will be pleased. This post-condition is an example of a domain-dependent inference rule, which is stored in the semantic network when a situation from the domain is discussed.

Finally, we have the rule memory. This is just the group of data objects whose names appear in the agenda. Unlike the other data objects, however, rules contain Lisp code, stored in two parts: the TEST and the ACTION. The TEST code is executed whenever the rule is being tried, and determines whether it fires or not. It is thus an indication of when this rule is applicable. (The conditions under which a rule is tried were given in the section on Control, section 2.1). The ACTION code is executed when the rule fires, and returns either a list of utterances (with an implied reply-wait), an utterance with an indication that no reply wait is necessary, or NIL, the standard Lisp symbol for "nothing". The rules can have side effects, such as creating a possible topic and then returning NIL. Although rules are connected into the topic-utterance graph, they are not really considered part of it, since they are a permanent part of the system, and contain Lisp code rather than CD expressions.

3. An example explained

A sample of what the present version of the system can do will now be examined. It is written in MacLisp, with utterances input and output in CD. This assumes the existence of programs to map English to CD and CD to English, both of which have been previously done to a degree. The agenda currently contains six rules. The two in the highest priority bucket stop the conversation if the other person says "goodbye" or leaves (Rule3-3 and Rule3-4). They are there to test the control of the system, and will have to be made more sophisticated (i.e., they should try to keep up the conversation if important active topics remain).

The three rules in the next bucket are the heart of the system at its current level of development. The first two raise topics to request missing information. The first (Rule1) asks about missing pre-conditions for a script instance, such as when someone who is not known to be a student takes a qualifier. The second (Rule2) asks about incompletely specified post-conditions, such as the actual project that someone must do if they get a remedial. At this university, a remedial is a conditional pass, where the student must complete a project in the same area as the qual in order to complete this degree requirement; there are four quals in the curriculum. The third rule in this bucket (Rule4) generates questions from topics that are open requests for information, and is illustrated in Figure 3-1.

```
RULE4
 TEST:  (FOR-EACH TOPICS
           (AND (EQUAL 'REQINFO (GET X
    'CPURPOSE))
              (NULL (GET X 'CLOSEDBY))))

 ACTION: (MAPCAN '(LAMBDA (X)
            (PROG (TMP)
              (RETURN (COND ((SETQ TMP
                          (QUESTIONIZE (GET-
    HYPO (
                   EVAL X))))
                     (MAPCAN '(LAMBDA (Y)
                           (COND (Y
          (LIST (UTTER Y (LIST X))))))
                          TMP))))))
          TEST-RESULT)
```

Test: Are there any topics which are requests for information which have not been answered?

Action: Retrieve the hypothetical part, form all "necessary" questions, and offer them as utterances.

Figure 3-1: Rule4

The last bucket in the agenda simply has a rule which says "I don't understand" in response to things that none of the previous rules generated a response to (RuleK). This serves as a safety net for the control structure, so it does not have to worry about what to do if no response is generated.

Now let us look at how the program handles an actual conversation fragment. The program always begins by asking "What's new?", to which (this time) it gets the reply, "Frank got a remedial on his hardware qual." The CD form for this is shown in Figure 3-2 (the program currently assumes that the person it is talking to is a student it knows named John). The CD version is an instance of the qual script, with Frank, hardware, and a remedial being the taker, area, and result, respectively.

```
U0002
((< = > ($QUAL &AREA (*HARDWARE*) &TAKER
   (*FRANK*) &RESULT (*REMEDIAL*))))
(ISA (*UTTERANCE*) PERSON *JOHN* PRED
   UTTS)
```

Figure 3-2: First input utterance

When the rules examine this, five topics are raised, one due to the pre-condition that he has not passed the qual before (by Rule1), and four due to various partially specified post-conditions (by Rule2):

- If Frank was confident, he will be unhappy.
- If he was not confident, he will be content.
- He has to do a project. We don't know what.
- If he has completed his project, he might be able to graduate.

The system only asks about things it does not know. In this case, it knows that Frank is a student, so it does not ask about that. As an example, the topic that asks whether he is content is illustrated in Figure 3-3.

```
T0005
((CON ((< = > ($QUAL &AREA
            (*HARDWARE*)
            &TAKER
            (*FRANK*)
            &RESULT
            (*REMEDIAL*))))
   LEADTO
   ((CON ((ACTOR (*FRANK*) IS
   (*CONFIDENCE* VAL (> 0)))
      MOD
      (*NEG* *HYPO*))
     LEADTO
     ((ACTOR (*FRANK*) IS (*HAPPINESS*
   VAL (0)))))
   MOD
   (*HYPO*))))
(INITIATED (U0013) SUCC T0009 CPURPOSE
   REQINFO
     INITIATEDBY (RULE2 U0002) ISA (*TOPIC*)
   PRED T0004)
```

Figure 3-3: A sample topic in detail

Along with raising these topics, the rules store the utterance and script post-inferences in the semantic network, under all the nodes mentioned in them. The following have been stored under Frank by this point:

- Frank got a remedial on his hardware qual.
- If he was confident, he'll be unhappy.
- If he was not confident, he'll be content.
- Passing the hardware qual will not contribute to his graduating.
- He has a hardware project to do.
- Finishing his hardware project will contribute to his graduating.

While these were being stored, Frank's goal tree was altered. This occurred because two of the post-inferences are themselves inference rules that affect whether he will graduate, and graduating is already assumed to be a goal of any student. Thus when the first is stored, a new goal tree is created for Frank (since his interests were represented before by the Student goal tree), and the goal of passing the hardware qual is removed. When the second is stored, the goal of finishing the project is added below that of graduating on Frank's tree. These goal trees are illustrated in Figures 3-4 and 3-5.

```
((ACTOR (*STUDENT*) IS (*HAPPINESS* VAL
   (5))))
                                      ) Subgoal
((< = > ($GRAD &ACTOR (*STUDENT*) &SCHOOL
   (*CMU*))))
                                      ) Subgoal
((< = > ($QUAL &TAKER (*STUDENT*) &AREA
   (*HARDWARE*) &RESULT (*PASSED*))))
```

Figure 3-4: A student's goal tree

```
((ACTOR (*FRANK*) IS (*HAPPINESS* VAL (5))))
                                      ) Subgoal
((< = > ($GRAD &ACTOR (*FRANK*) &SCHOOL
   (*CMU*))))
                                      ) Subgoal
((< = > ($PROJECT &STUDENT (*FRANK*) &AREA
   (*HARDWARE*) &RESULT (*COMPLETED*)))
    MOD (*HYPO*) TIME (> *NOW*))
```

Figure 3-5: Frank's new goal tree

At this point, six utterances are generated by Rule4. They are given in Figure 3-6. Three are generated from the first topic, one is generated from each of the next three topics, and none is generated from the last topic. The interest rating routine now compares these utterances to Frank's goals, and picks the most interesting one. Because of the new goal tree, the last three utterances match none of Frank's goals, and receive zero ratings. The first one matches his third goal in a neutral way, and receives a rating of 56 (an utterance receives 64 points for the top goal, minus 4 for each level below top, plus or minus one for positive/negative implications. These numbers are, of course, arbitrary, as long as ratings from different goals do not overlap). The second one matches his top goal in a neutral way, and receives 64. Finally, the third one matches his top goal in a negative way, and receives 63. Therefore, the second question gets uttered, and ends up with the links shown in Figure 3-7. The other generated utterances are discarded, possibly to be regenerated later, if their topics are still open.

```
((< = > ($PROJECT &STUDENT (*FRANK*) &AREA
   (*HARDWARE*) &BODY (*?*))))
What project does he have to do?

((ACTOR (*FRANK*) IS (*HAPPINESS* VAL (0)))
   MOD (*?*))
Is he content?

((ACTOR (*FRANK*) IS (*HAPPINESS* VAL (-3)))
   MOD (*?*))
Is he unhappy?

((< = > ($QUAL &TAKER (*FRANK*) &AREA
   (*HARDWARE*))) MOD (*?* *NEG*))
Hadn't he taken it before?

((< = > ($QUAL &TAKER (*FRANK*) &AREA
   (*HARDWARE*) &RESULT (*CANCELLED*)))
   MOD (*?*))
Had it been cancelled on him before?

((< = > ($QUAL &TAKER (*FRANK*) &AREA
   (*HARDWARE*) &RESULT (*FAILED*))) MOD
   (*?*))
Had he failed it before?
```

Figure 3-6: The six possible utterances generated

4. Other work, future work

Two other approaches used in modelling conversation are task-oriented and speech acts based systems. Both of these methodologies have their merits, but neither attacks all the same aspects of the problem that this system does. Task-

```
U0013
((ACTOR (*FRANK*) IS (*HAPPINESS* VAL (0)))
    MOD (*?*))
(PRED U0002 ISA (*UTTERANCE*) PERSON
    *ME*
    INTEREST-REASON (G0006) INTEREST 64
    INITIATEDBY (RULE4 T0005))
```

Figure 3-7: System's response to first utterance

oriented systems [5] operate in the context of some fixed task which both speakers are trying to accomplish. Because of this, they can infer the topics that are likely to be discussed from the semantic structure of the task. For example, a task-oriented system talking about qualifiers would use the knowledge of how to be a student in order to talk about those things relevant to passing qualifiers (simulating a very studious student). It would not usually ask a question like "Is Frank content?", because that does not matter from a practical point of view.

Speech acts based systems (such as [1]) try to reason about the plans that the actors in the conversation are trying to execute, viewing each utterance as an operator on the environment. Consequently, they are concerned mostly about what people mean when they use indirect speech acts (such as using "It's cold in here" to say "Close the window") and are not as concerned about trying to say *interesting* things as this system is. Another way to look at the two kinds of systems is that speech acts systems reason about the actors' plans and assume fixed goals, whereas this system reasons primarily about their goals.

As for related work, ELI (the language analyzer mentioned in section 1) and this system (when fully developed) could theoretically be merged into a single conversation system, with some rules working on mapping English into CD, and others using the CD to decide what responses to generate. In fact, there are situations in which one needs to make use of both kinds of information (such as when a phrase signals a topic shift: "On the other hand..."). One of the possible directions for future work is the incorporation and integration of a rule-based parser into the system, along with some form of rule-based English generation. Another related system, MICS [3], had research goals and a set of knowledge sources somewhat similar to this system's, but it differed primarily in that it could not alter its goal trees during a conversation, nor did it have explicit data structures for representing topics (the selection of topics was built into the interpreter).

The main results of this research so far have been the topic-utterance graph and dynamic goal trees. Although some way of holding the intersentential information was obviously needed, no precise form was postulated initially. The current structure was invented after working with an earlier set of rules to discover the most useful form the topics could take. Similarly, the idea that a changing view of someone else's goals should be used to control the course of the conversation arose during work on producing the interest-rating routine. The current system is, of course, by no means a complete model of human discourse. More rules need to be developed, and the current ones need to be refined.

In addition to implementing more rules and incorporating a parser, possible areas for future work include replacing the interest-rater with a second agenda (containing interest-determining rules), changing scripts and testing whether the rules are truly independent of the subject matter, trying to make the system work with several scripts at once (as SAM [4] does), and improving the semantic network to handle the well-known problems which may arise.

References

[1] Allen, J. F. and Perrault, C. R.
Analyzing Intention in Utterances.
Artificial Intelligence 15(3):143-178, December, 1980.

[2] Brachman, R. J.
On the Epistemological Status of Semantic Networks.
In Findler, N. V. (editor), *Associative Networks: Representation and Use of Knowledge by Computers*, chapter 1 in particular. Academic Press, New York, 1979.

[3] Carbonell, J. G.
Subjective Understanding: Computer Models of Belief Systems.
PhD thesis, Yale University, January, 1979.
Computer Science Research Report #150.

[4] Cullingford, R. E.
Script Application: Computer Understanding of Newspaper Stories.
PhD thesis, Yale University, January, 1978.
Computer Science Research Report #116.

[5] Grosz, B.J.
The Representation and use of Focus in Dialogue Understanding.
Technical Report 151, Stanford Research Institute, July, 1977.

[6] Newell, A. and Simon, H. A.
Human Problem Solving.
Prentice Hall, Englewood Cliffs, N. J., 1972, chapter 8.

[7] Riesbeck, C. and Schank, R. C.
Comprehension by Computer: Expectation Based Analysis of Sentences in Context.
Technical Report 78, Department of Computer Science, Yale University, 1976.

[8] Schank, R. C.
Conceptual Information Processing.
North-Holland, 1975, chapter 3.

[9] Schank, R. C. and Abelson, R.
Scripts, Plans, Goals and Understanding.
Erlbaum, 1977, chapter 3.