Mumbling - User-Driven Cooperative Interaction

by

Jussi Karlsgren
Mumbling - User-Driven Cooperative Interaction

BY

Jussi Karlgren

NLP-Group; January 21, 1994

jussi@sics.se

Swedish Institute of Computer Science
Box 1263, S-164 28 KISTA, SWEDEN

ABSTRACT

This paper suggests a scheme for raising the cooperativeness of natural language interface without changing either modality or system linguistic competence, but by heightening the level of interactivity and by aiding the user in maintaining the responsibility for the discourse. In short: hands-off-pragmatics at the computer interface.

KEYWORDS:
Natural Language Interface, Discourse, Pragmatics, Adaption, Convergence
Mumbling – User-Driven Cooperative Interaction

Jussi Karlsgren, jussi@sics.se

January 21, 1994

Abstract

This paper suggests a scheme for raising the cooperativeness of natural language interfaces without changing either modality or system linguistic competence, but by heightening the level of interactivity and by aiding the user in maintaining the responsibility for the discourse. In short: hands-off-pragmatics at the computer interface.

Introduction

The idea of interacting with computers in “natural language” seems to be excellent, but has not caught on. There is a number of serious drawbacks with natural language interfaces as they most often have been presented, and they most often invite critics to draw the wrong conclusions. The naïve view of natural language interaction is that it will enable users to communicate with computers without training. This is easy to disprove, which has been done. (Shneiderman, 1980). Natural language systems are opaque to the user: what language they actually handle is not obvious. The users need to be coached into using the language variety the system handles. One of the main problems of designing natural language systems should be how to teach the user the variety at hand (Rich, 1980). Systems today generally are uncooperative in this respect.

The discussion is most often found in papers that suggest direct manipulation or some other non-textual mode of communication as a way of overcoming the drawbacks (Bretan, 1990, 1992; Cohen, 1992; Bretan and Karlsgren, 1993). In this paper the focus is on what can be done on a high level of pragmatics without changing either modality or system linguistic competence to effect a more cooperative interaction.

One-Shot Interaction

The underlying problem of natural language interfaces is the problem of “one-shot”-interaction: systems expect users to pose queries in one go, rather than discuss a topic until consensus is reached. This is not a natural way of using
natural language: the dialog model is highly artificial and difficult to learn to use. The whole premise, that a user would be able to, without help, frame one single query with a well-defined content in terms of the system knowledge base is alien to the nature of natural language — one-shot dialogs are rare. The dialog model appears to be conversational, but does not support conversation. The crucial point is that natural dialog is not only interactive but also incremental — a structure which human-computer dialog in today’s systems does not support. In conversation, both parties take the responsibility to maintain this incrementality. With one party a computer, the human user will not expect it to be able to. The computer should let the user take full responsibility, but aid the user in doing so. The two problems are related on the level of actual use. If the linguistic competence of the system is difficult to determine, the dialog model should cope with retries and support discussion.

The linguistic competence users expect from computers is extremely simple, which has been shown in a number of studies (Malhotra, 1975; Thompson, 1980; Wachtel, 1986; Guindon 1987; Kennedy et al 1988). This is specifically true for discourse structure, which Nils Dahlbäck and Arne Jönsson have shown can be modelled by an exceedingly simple dialog grammar, by examining the discourse structure in the material obtained by carefully designed Wizard of Oz studies (Dahlbäck, 1991; Dahlbäck and Jönsson, 1986; Dahlbäck, Jönsson, and Ahrenberg, 1993). This can be explained by a fundamental asymmetry of beliefs between user and system as posed by Aravind Joshi (1982). Users do not expect computer systems to take responsibility for the coherency of a discourse, but expect to take full responsibility for the discourse management themselves.

This fact, together with the natural tendency of users to mimic their counterparts in discourse (Ray and Webb, 1966; Levelt and Kelter 1982; Isaacs and Clark, 1987; Krauss and Fussell, 1992), can be exploited in the design of interactive systems (Brennan, 1991; Zoltan-Ford, 1991; Karlgren 1992).

In short, the problem is non-transparency of the system competence. Also in short, the solution is to use natural mechanisms of the user to have the user learn the system competence. If the system produces the kind of language it understands, the users will pick it up.

Leaving the Responsibility to the User

In any interactive situation today, users will typically expect the system to ram its knowledge representation down their throats. This expectation can be utilized to aid user interaction. The system will have to produce such information to the user that will support user decision making by displaying as much as possible of the system knowledge structures. This does not have to be done in an unfriendly way.

---

1 As pointed out by Victor Zue
This is still just a one-shot from the system’s point of view – taking context into account is difficult, and best left to the user. What the system does in this scenario is simply to leave the responsibility to maintain dialog structure to its users, and as an aid to the users, give them a good basis for making decisions about where to go.

The point is that however sophisticated the system – and the toy system in the examples below certainly isn’t sophisticated at all – there will always be situations where the system will have no competence to decide. This when the principles outlined below will be useful. And, with the state-of-the-art of natural language interaction to-day, this approach is simply necessary.

This idea of helping the user mimic system competence has been used in IBM SAA LanguageAccess (Sanamrad and Bretan, 1992), for its Model Help Component, which when users posed queries about a term in the lexicon output sentences in which the term was put to use, as a way of familiarizing the user with the content of the conceptual model and of the coverage of the grammar (Petrović, 1992).

Given a well formalized model of domain knowledge users will expect a computer to act like a computer. (A simple entity-relationship model of knowledge is assumed here (Chen, 1976), implemented in Prolog (Johannesson, 1990), although any approach with an explicit domain knowledge representation would be just as reasonable.) Users will not be surprised if the system is inflexible, but they will want to know what the system knows, or in this case, what the conceptual model contains.

In short, display what is difficult, using language that the system understands, and that the user thus can recycle.

The Simplest Pragmatics Component Conceivable

The interaction is processed as indicated in Figure 1. A user query is parsed, resulting in a semantic representation. The representation of the query is sent to be further processed by the back-end system. The mumble component mumbles about synonyms of recognized terms, about entities and relations recognized in the query, and displays the relations and entities recognized prominently, for user consideration. If the system does not understand the representation, the mumble component will mumble generally about the different possible themes the system expects. All textual mumbling will be done using the same grammar the analysis component accepts.

This system layout is identical irrespective of the degree of sophistication or other characteristics of the analysis, interaction format, or back-end process at hand. Indeed, it is not dependent on the modality of the interaction. If a visual modality is available, it will be immensely useful within the same framework (Bretan and Karlgren, 1993).

This is not plan inference. The system will not attempt to guess or infer what the user intends, or what error may have been made. It simply will output
whatever it can, about whatever it recognizes in the interaction.

This is not unrelated to the commenting as proposed and implemented in the Edward system by Claassen et al (1993). Edward gives continuous verbal feedback on objects that are manipulated on screen in a direct manipulation interface to an operating system.

An Example Interface

The Analysis Component

In the following example a very simple sentence linguistic analysis component is used: it parses the string and assembles a representation of the semantic content by simply collecting all keywords it recognizes in a list. Any nicer module can be plugged in here².

The Back-End System

The system knowledge of itself is represented in the knowledge base. Again, in this example implementation, the representations are exceedingly simple: keywords are related to goals, and the representation of the goal is the collection of keywords that are related to it.

In an interaction, the semantic representations of the queries are matched to representations of goals in the system knowledge base.

The Display Component

In this example, simple canned text is output to text terminal.

²Experiments are under way.
The Mumble Component

The mumble component in this minimal implementation always identifies all conceivable entities, relations, and entity sets in the query and mentions them using constructions that would pass through the analysis component.

To show the usefulness of the mumbling mechanism it is not necessary to implement more complex error critiquing, plan libraries or the like – the essential qualities will be the same, even if the actual implementation will vary with the sophistication of the analysis. With a more sophisticated model of interaction and task, the same process takes place, only with finer granularity.

An Example Interaction

Example 1

The query

What is the full name of Joe?
will have the mumble component produce
This is about the attribute full_name.
A person has a full_name.
joe is a man.
A man is a person.
A person is the fellow of a membership.
A person has a full_name.
A man is the husband of a marriage.

Assuming the system will be able to produce output from this, (which it ought to, given the conceptual model shown above) this may not seem all that interesting, except for the fact that the user will be enlightened to what relationships a man can enter into.

Example 2

In the following example the user poses a query that cannot be answered by the system as it stands.

Who is the wife of Pete?
will have the mumble component produce
This is about the attribute wife.
A woman is the wife of a marriage.
Pete is a man.
A man is a person.
A person is the fellow of a membership.
A person has a full_name.
A man is the husband of a marriage.

which should lead the user to ask something like
What is Pete's marriage?
and then
Who is the wife of marriage42?

Example 3

What is Joe's club?
will have the mumble component produce
Joe is a man.
A man is a person.
A man is the husband in a marriage.
A person has a name.
A person is a fellow in a membership.
A club has a name.
A club is an organization in a membership.

which should lead the user to ask something like
What is Joe's membership?
and then
What is the club of membership?

---

3Yes, the conceptual model is silly.
How To Mumble

As one example of how mumbling can be implemented the following scheme gives results as shown in the preceding sections. Naturally several more complex schemes could prove useful as well — this is a minimal model.

Given the conceptual model in Figure 3 where entity(x) is an instance of entity type "a" the following simple cases of mumbling can be necessary. Mumbling on

- entities such as "x",
- entity types such as "a", "b" and so on,
- relations of different kinds such as "d".

We have in the examples in the preceding section seen how mumbling can be useful. Again, this will be dependent on the exact nature of the knowledge representation used, but when a user asks about the "f" of x, where "f"s and "a"s are not linked a reasonable strategy is to mumble on both "f"s and "a"s, and to explain that x is an "a". Example 2 in the preceding section is an example of this, where a user asks the system about Pete's wife.

In a case like this, the system should mumble by mentioning what x is ("x is an "a".") and continuing by mumbling on both entity types involved, mentioning relations the entity types enter into ("An "a" is the "d" in a "c".") or hypernyms ("An "a" is a "b".") as well as datatypes ("x has an "e"."). This is what the mumbling mechanism does in the examples shown.

Scaling Up

The interaction in a purely verbal form as above will explode if the conceptual model is more complex, or even simply if the entities have more attributes. The number of phrases generated is one for each attribute an entity has, in the model shown.
This needs to be remedied. One way to remedy this would be to add complex path searching algorithms to search the conceptual model intelligently to the mumble component, or to either add knowledge representation to the component or to couple it closer with the background system. This, however, would be against the idea of mumbling: the mumbling mechanism is a part of a more general discourse management framework, not of sensible design of knowledge representation. Besides, firstly, the current ideology of knowledge representation is to keep the the formalism for representation as simple and pure as possible, and secondly, whatever the knowledge representation, it will never be specialized enough to handle every task in every situation in a domain, nor will it be general enough to cover all domains without trouble. There will always be mismatches between user knowledge and system knowledge. If the representation is complex, designing a mumbling component will involve more work, and hopefully result in more exact mumbling.

The future direction in which mumbling will be reduced to manageable proportions is to use discourse knowledge in that the idea is to attempt identify references to entities, relations, and entity sets, to build up a discourse history, and in the longer term to build up an analysis of the topic of discourse. This, in general, is a very difficult question. However, in the interactive systems of today, the general problem does not need to be solved. A well defined problem area is given when the user sits down to communicate with the system, and a well defined conceptual model is accessible to refer to. Given these premises, the problem of topic identification shows promise to be reducible to manageable proportions.

These approaches will be studied and put to use in a number of different projects at SICS in the near future.
References


Appendix: Code For The Mumbler

/*
mumble.pl
==================================
Jussi Karlsgren, Saltsjo-Boo
1993
Experiment program for
broader interaction
==================================
*/

:- dynamic person/1.

g :-
getstring(S),
parse(S,Sem,Terms),
process(Sem,Answer),
numble(Terms,Context),
output(Answer),
output(Context).

%mumble on items in K until
%list is exhausted.
mumble(K,L) :- mumbledl(K,L,[ ]).

%----------------------------
%List of items is empty
%mumbledl([],X,X).

%K is an entity type.
mumbledl([K|Ks],[L|L],[ ]):-
entity(K),
Lj = ["There","is","a","K","."],
entumble(K,L,L),
mumbledl(Ks,L,L).

%K is an entity instance.
%Find the most specific
%entity type and mention it.
mumbledl([K|Ks],[L|L],Ls) :-
entity(Class),
G =..[Class,K],
call(G),
\+ hyponym(K,Class),
Lj = ["K","is","a","Class","."],
entumble(Class,L,L),
mumbledl(Ks,L1,Ls).

hyponym(K,Class) :-
isa(SubClass,Class),
entity(SubClass),
G =..[Class,K],
call(G).

%K is a relation.
mumbledl([K|Ks],L,Ls) :-
\+ attribute(K,_,_),
rnumble(K,L,L1),
mumbledl(Ks,L1,Ls).

%mumble on entities.
%1. Hyponyms.
%2. Relations where K is domain.
%3. Relations where K is range.

%----------------------------
%Is a relation.

\+ attribute(K,_,_),
\+ entity(K),
entity(Class),
G =..[Class,K],
\+ call(G),
mumbledl(Ks,L,L).

%----------------------------
%Is the most specific.

\+ isaumle(K,L,L),
inumbleLHS(K,L,L),
eumbleLHS(K,L,L).

%----------------------------
%Is the most specific.

\+ isamnumle(EntClass,L,L),
iseumble(EntClass,[ISA|L],L1) :-
entity(EntClass),
isu(EntClass,SuperEntClass),
ISA = ["K","EntClass","is","a",SuperEntClass,"."],
eumbleSuperEntClass(L,L).

%----------------------------
%Is the most specific.

\+ eumbleRHS(EntClass,L,L),
findall(EntClass-G-D,
attribute(Q,domai(D),
range(EntClass),
maping(i,m,t,p)),
Ents),
sents(Ents,L,L).
enumbleLHS(EntClass,L,Lr) :-
findall(R-Q-EntClass,
    attribute(Q,
        domain(EntClass),
        range(R),
        mapping(1,m,t,p)),
    Ents),
sents(Ents,L,Lr).

%------------------------------------------------------
% Mumble on relation Q.
% Mention all entities that
% enter into Q with each other.
rmumble(Q,[L,L0],L) :-
    L = ["This","is","about","the",
        "attribute",Q,"."],
     findall(R-Q-EntClass,
        attribute(Q,
            domain(EntClass),
            range(R),
            mapping(1,m,t,p)),
        Ents),
sents(Ents,L0,L).

%------------------------------------------------------
% Generate sentence for relation.
% 1. Relation between entities
% 2. Data type.
sents([],X,X).
sents([A-Q-D|Ents],[S|L1],L2) :-
    entity(A),
    S = ["A",A,"is","the",Q,
        "of","a",D,"."],
sents(Ents,L1,L2).

sents([A-Q-D|Ents],[S|L1],L2) :-
    data_type(A),
    S = ["A",D,"has","a",Q,"."],
sents(Ents,L1,L2).
Appendix: Conceptual Model and Information Base

The conceptual model and information base are an abbreviated version of an example provided by Paul Johannes-son and Magnus Boman.

/* Conceptual model */
entity(person).
attribute(full_name, domain(person), range(p_name), mapping(1,m,t,p)).

entity(man). isa(man,person). person(X) :- man(X).

entity(woman). isa(woman,person). person(X) :- woman(X).

entity(marriage). attribute(husband, domain(marriage), range(man), mapping(1,m,t,p)).
attribute(wife, domain(marriage), range(woman), mapping(1,m,t,p)).

entity(club). attribute(club_name, domain(club), range(text), mapping(1,1,t,p)).

entity(membership). attribute(organization, domain(membership), range(club), mapping(1,m,t,p)).
attribute(fellow, domain(membership), range(person), mapping(1,m,t,p)).