NATURAL LANGUAGE DATABASE UPDATES

Sharon C. Salveter
David Maier
Computer Science Department
SUNY Stony Brook
Stony Brook, NY 11794

ABSTRACT

Although a great deal of research effort has been expended in support of natural language (NL) database querying, little effort has gone to NL database update. One reason for this state of affairs is that in NL querying, one can tie nouns and stative verbs in the query to database objects (relation names, attributes and domain values). In many cases this correspondence seems sufficient to interpret NL queries. NL update seems to require database counterparts for active verbs, such as "hire," "schedule" and "enroll," rather than for stative entities. There seem to be no natural candidates to fill this role.

We suggest a database counterpart for active verbs, which we call verbgraphs. The verbgraphs may be used to support NL update. A verbgraph is a structure for representing the various database changes that a given verb might describe. In addition to describing the variants of a verb, they may be used to disambiguate the update command. Other possible uses of verbgraphs include, specification of defaults, prompting of the user to guide but not dictate user interaction and enforcing a variety of types of database integrity constraints.

I. MOTIVIATION AND PROBLEM STATEMENT

We want to support natural language interface for all aspects of database manipulation. English and English-like query systems already exist, such as ROBOT[Ha77], TQA[Da78], LUNAR[WO76] and those described by Kaplan[Ka79], Walker[Wa78] and Waltz [Wz75]. We propose to extend natural language interaction to include data modification (insert, delete, modify) rather than simply data extraction. The desirability and unavailability of natural language database modification has been noted by Wiederhold, et al.[Wi81]. Database systems currently do not contain structures for explicit modelling of real world changes.

A state of a database (DB) is meant to represent a state of a portion of the real world. We refer to the abstract description of the portion of the real world being modelled as the semantic data description (SDD). A SDD indicates a set of real world states (RWS) of interest, a DB definition gives a set of allowable database states (DBS). The correspondence between the SDD and the DB definition induces connections between DB states and real world states. The situation is diagrammed in Figure 1.

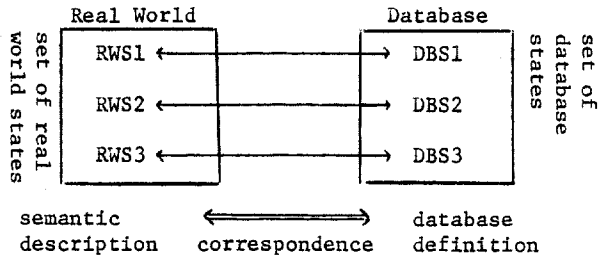

Figure 1

Natural language (NL) querying of the DB requires that the correspondence between the SDD and the DB definition be explicitly stated. The query system must translate a question phrased in terms of the SDD into a question phrased in terms of a data retrieval command in the language of the DB system. The response to the command must be translated back into terms of the SDD, which yields information about the real world state. For NL database modification, this stative correspondence between DB states and real world states is not adequate. We want changes in the real world to be reflected in the DB. In Figure 2 we see that when some action in the real world causes a state change from RWS1 to RWS2, we must perform some modification to the DB to change its state from DBS1 to DBS2.

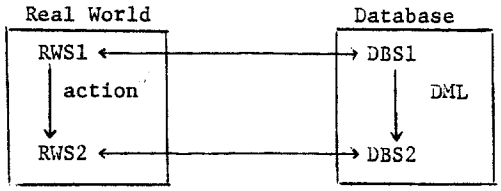

Figure 2

This research is partially supported by NSF grants IST-79-18264 and ENG-79-07794.

We have a means to describe the action that changed the state of the real world: active verbs. We also have a means to describe a change in the DB state: data manipulation language (DML) command sequences. But given a real world-action, how do we find a DML command sequence that will accomplish the corresponding change in the DB?

Before we explore ways to represent his active correspondence--the connection between real world actions and DB updates--, let us examine how the stative correspondence is captured for use by a NL query system. We need to connect entities and relationships in the SDD with files, fields and field values in the DB. This stative correspondence between RWS and DBS is generally specified in a system file. For example, in Harris' ROBOT system, the semantic description is implicit, and it is assumed to be given in English. The entities and relationships in the description are roughly English nouns and stative verbs. The correspondence of the SDD to the DB is given by a lexicon that associates English words with files, fields and field values in the DB. This lexicon also gives possible referents for word and phrases such as "who," "where" and "how much."

Consider the following example. Suppose we have an office DB of employees and their scheduled meetings, reservations for meeting rooms and messages from one employee to another. We capture this information in the following four relations,

```
EMP(name,office,phone,supervisor)
APPOINTMENT(name,date,time,duration,who,
            topic,location)
MAILBOX(name,date,time,from,message)
ROOMRESERVE(room,date,time,duration,reserver)
```

with domains (permissible sets of values):

DOMAIN	ATTRIBUTES
personname	name, who, from, reserver, supervisor
roomnum	room, location, office
phonenum	phone
calendardate	date
clocktime	time
elapsedtime	duration
text	message, topic

Consider an analysis of the query

> "What is the name and phone # of the person who reserved room 85 for 2:45pm today?"

Using the lexicon, we can tie words in the query to domains and relations.

name - personname	reserve - ROOMRESERVE relation
phone - phonenum	room - roomnum
person - personname	2:45pm - clocktime
who - personname	today - calendardate

We need to connect relations EMP and ROOMRESERVE. The possible joins are room-office and name-reserver. If we have stored the information that offices and reservable rooms never intersect, we can eliminate the first possibility. Thus we can arrive at the query

> in EMP, ROOMRESERVE retrieve name, phone where name = reserver and room = 85 and time = 2:45pm and date = CURRENTDATE

Suppose we now want to make a change to the database:

> "Schedule Bob Marley for 2:15pm Friday."

This request could mean schedule a meeting with an individual or schedule Bob Marley for a seminar. We want to connect "schedule" with the insertion of a tuple in either APPOINTMENT or ROOMRESERVE. Although we may have pointers from "schedule" to APPOINTMENT and ROOMRESERVE, we do not have adequate information for choosing the relation to update.

Although files, fields, domains and values seem to be adequate for expressing the stative correspondence, we have no similar DB objects to which we may tie verbs that describe actions in the real world. The best we can do with files, fields and domains is to indicate what is to be modified; we cannot specify how to make the modification. We need to connect the verbs "schedule," "hire" and "reserve" with some structures that dictate appropriate DML sequences that perform the corresponding updates to the DB. The best we have is a specific DML command sequence, a transaction, for each instance of "schedule" in the real world. No single transaction truly represents all the implications and variants of the "schedule" action. "Schedule" really corresponds to a set of similar transactions, or perhaps some parameterized version of a DB transaction.

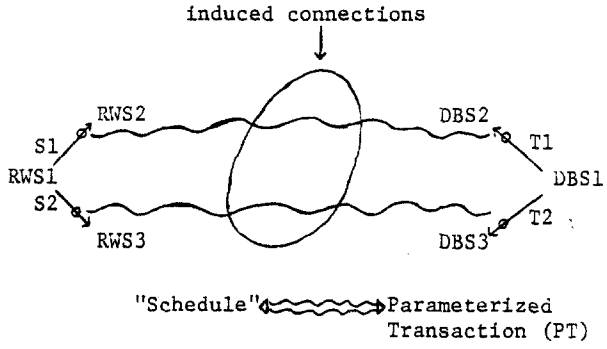

Figure 3

The desired situation is shown in Figure 3. We have an active correspondence between "schedule" and a parameterized DB transaction PT. Different instances of the schedule action, S1 and S2, cause different changes in the real world state. From the active correspondence of "schedule" and PT, we want to produce the proper transaction, T1 or T2, to effect the correct change in the DB state. There is not an existing candidate for the high-level specification language for verb descriptions.

We must be able to readily express the correspondence between actions in the semantic world and verb descriptions in this high-level specification We depend heavily on this correspondence to process natural language updates, just as the stative correspondence is used to process natural language queries. In the next section we examine these requirements in more detail and offer, by example, one candidate for the representation.

Another indication of the problem of active verbs in DB shows up in looking a semantic data languages. Sematnic data models are systems for constructing precise descriptions of protions of the real world - semantic data description (SDD)- using terms that come from the real world rather than a particular DB system. A SDD is a starting point for designing and comparing particular DB implementations. Some of the semantic models that have been proposed are the entity-relationship model[Ch76], SDM[HM81], RM/T[Co79], TAXIS[MB80] and Beta[Br78]. For some of these models, methodologies exist for translating to a DB specification in various DB models, as well as for expressing the static correspondence between a SDD in the semantic model and a particular DB implementation. To express actions in these models, however, there are only terms that refer to DBs: insert, delete, modify, rather than schedule, cancel, postpone (the notable exception is Skuce[Sk80]).

While there have been a number of approaches made to NL querying, there seems to be little work on NL update. Carbonell and Hayes[CH81] have looked at parsing a limited set of NL update commands, but they do not say much about generating the DB transactions for these commands. Kaplan and Davidson[KD81] have looked at the translation of NL updates to transactions, but the active verbs they deal with are synonyms for DB terms, essentially following the semantic data model as above. This limitation is intentional, as the following excerpt shows:

> First, it is assume that the underlying database update must be a series of transactions of the same type indicated in the request. That is, if the update requests a deletion, this can only be mapped into a series of deletions in the database.

While some active verbs, such as "schedule," may correspond to a single type of DB update, there are other verbs that will require multiple types of DB updates, such as "cancel," which might require sending message as well as removing an appointment. Kaplan and Davidson are also trying to be domain independent, while we are trying to exploit domain-specific information.

II. NATURE OF THE REPRESENTATION

We propose a structure, a verbgraph, to represent action verbs. Verbgraph are extensions of frame-like structures used to represent verb meaning in MORAN[Sa78] and [Sa79]. One verbgraph is associated with each sense of a verb; that structure represents all variants. A real world change is described by a sentence that contains an active verb; the DB changes are accomplished by DML command sequences. A verbgraph is used to select DML sequences appropriate to process the variants of verb sense. We also wish to capture that one verb that may be used as part of another: we may have a verb sense RESERVE-ROOM that may be used by itself or may be used as a subpart of the verb SCHEDULE-TALK.

Figure 4 is an example of verbgraph. It models the "schedule appointment" sense of the verb "schedule." There are four basic variants we are attempting to capture; they are distinguished by whether or not the appointment is scheduled with someone in the company and whether or not a meeting room is to be reserved. There is also the possibility that the supervisor must be notified of the meeting.

The verbgraph is directed acyclic graph (DAG) with 5 kinds of nodes: header, footer, information, AND (⊛) and OR (⊙). Header is the source of the graph, the footer is the sink. Every information node has one incoming and outgoing edge. An AND or OR node can have any number of incoming or outgoing edges. A variant corresponds to a directed path in the graph. We define a path to be connected subgraph such that

1) the header is included;
2) the footer is included;
3) if it contains an information node, it contains the incoming and outgoing edge;
4) if it contains an AND node, it contains all incoming and outgoing edges; and
5) if it contains an OR node, it contains exactly one incoming and one outgoing edge.

We can think of tracing a path in the graph by starting at the header and following its outgoing edge. Whenever we encounter an information node, we go through it. Whenever we encounter an AND node, the path divides and follows all outgoing edges. We may only pass through an AND node if all its incoming edges have been followed. An OR node can be entered on only one edge and we leave it by any of its outgoing edges.

An example of a complete path is one that consists of theheader, footer, information nodes, A, B, D, J, and connector nodes, a, b, c, d, g, k, l, n. Although there is a direction to paths, we do not intend that the order of nodes on a path implies any order of processing the graph, except the footer node is always last to be processed. A variant of a verb sense is described by the set of all expressions in the information nodes contained in a path.

Expressions in the information nodes can be of two basic types: assignment and restriction. The assignment type produces a value to be used in the update, either by input or computation; the key word input indicates the value comes from the user. Some examples of assignment are:

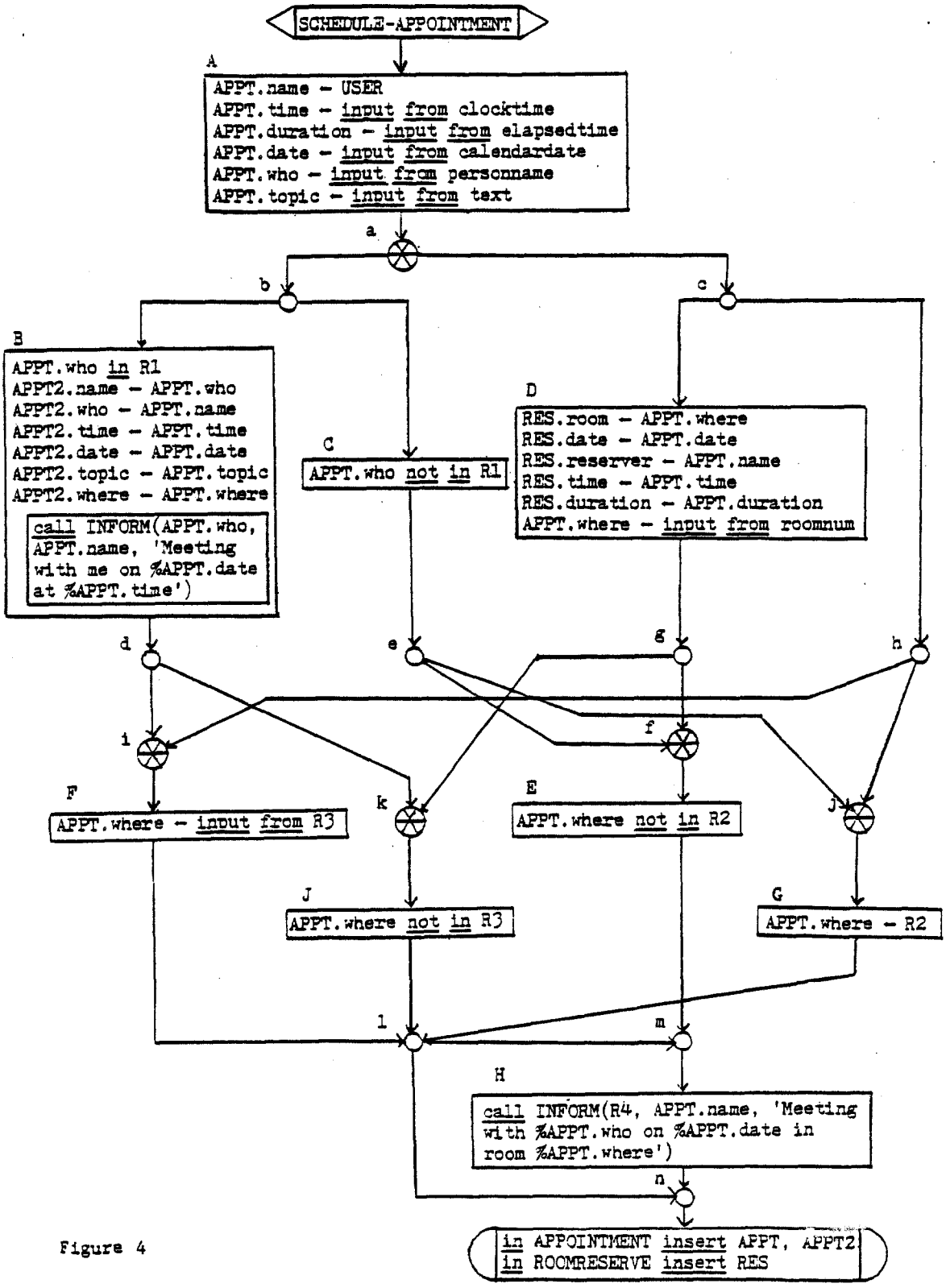
SCHEDULE-APPOINTMENT
A
APPT.name - USER
APPT.time - input from clocktime
APPT.duration - input from elapsedtime
APPT.date - input from calendardate
APPT.who - input from personname
APPT.topic - input from text
a
b
c
B
APPT.who in R1
APPT2.name - APPT.who
APPT2.who - APPT.name
APPT2.time - APPT.time
APPT2.date - APPT.date
APPT2.topic - APPT.topic
APPT2.where - APPT.where
call INFORM(APPT.who, APPT.name, 'Meeting with me on %APPT.date at %APPT.time')
C
APPT.who not in R1
D
RES.room - APPT.where
RES.date - APPT.date
RES.reserver - APPT.name
RES.time - APPT.time
RES.duration - APPT.duration
APPT.where - input from roomnum
d
e
g
h
i
f
F
APPT.where - input from R3
k
E
APPT.where not in R2
j
J
APPT.where not in R3
G
APPT.where - R2
l
m
H
call INFORM(R4, APPT.name, 'Meeting with %APPT.who on %APPT.date in room %APPT.where')
n
in APPOINTMENT insert APPT, APPT2
in ROOMRESERVE insert RES

Figure 4

1) (node labelled A in Figure 4)
APPT.who ← input from personname

The user must provide a value from the domain personname.

2) (node labelled D in Figure 4)
RES.date ← APPT.date

The value for APPT.date is used as the value RES.date.

An example of restriction is: (node B in Figure 4)
APPT.who in R1 where R1 = in EMP retrieve name

This statement restricts the value of APPT.who to be a company employee.

Also in Figure 4, the symbols R_1, R_2, R_3 and R_4 stand for the retrievals

R_1 = in EMP retrieve name

R_2 = in EMP retrieve office where name = APPT.name

R_3 = in EMP retrieve office where name = APPT.name or name = APPT.who.

R_4 = in EMP retrieve supervisor where name = APPT.name.

In Node B, INFORM(APPT.who, APPT.name, 'meeting with me on %APPT.date at %APPT.time') stands for another verbgraph that represents sending a message by inserting a tuple in MAILBOX. We can treat the INFORM verbgraph as a procedure by specifying values for all the slots that must be filled from input. The input slots for INFORM are (name, from, message).

III. WHAT CAN WE DO WITH IT?

One use for the verbgraphs is in support of NL directed manipulation of the DB. In particular, they can aid in variant selection. We assume that the correct verb sense has already been selected; we discuss sense selection later. Our goal is to use information in the query and user responses to questions to identify a path in the verbgraph. Let us refer again to the verbgraph for SCHEDULE-APPOINTMENT shown in Figure 4. Suppose the user command is "Schedule an appointment with James Parker on April 13" where James Parker is a company employee. Interaction with the verbgraph proceeds as follows. First, information is extracted from the command and classified by domain. For example, James Parker is in domain personname, which can only be used to instantiate APPT.name, APPT.who, APPT2.name and APPT2.who. However, since USER is a system variable, the only slots left are APPT.who and APPT2.name, which are necessarily the same. Thus we can instantiate APPT.who and APPT2.name with "James Parker." We classify "April 13" as a calendar date and instantiate APPT.date, APPT2.date and RES.date with it, because all these must be the same. No more useful information is in the query. Second, we examine the graph to see if a unique path has been determined. In this case it has not. However, other possibilities are constrained because we know the path must go through node B. This is because the path must go through either node B or node C and by analyzing the response to retrieval R1, we can determine it must be node B (i.e., James Parker is a company employee). Now we must determine the rest of the path. One determination yet to be made is whether or not node D is in the path. Because no room was mentioned in the query, we generate from the graph a question such as "Where will the appointment take place?" Suppose the answer is "my office." Presume we can translate "my office" into the scheduler's office number. This response has two effects. First, we know that no room has to be reserved, so node D is not in the path. Second, we can fill in APPT.where in node F. Finally, all that remains to be decided is if node H is on the path. A question like "Should we notify your supervisor?" is generated. Supposing the answer is "no." Now the path is completely determined; it contains nodes A, B and F. Now that we have determined a unique path in the graph, we discover that not all the information has been filled-in in every node on the path. We now ask the questions to complete these nodes, such as "What time?", "For how long?" and "What is the topic?". At this point we have a complete unique path, so the appropriate calls to INFORM can be made and the parameterized transaction in the footer can be filled-in.

Note that the above interaction was quite rigidly structured. In particular, after the user issues the original command, the verbgraph instantiation program chooses the order of the subsequent data entry. There is no provision for default, or optional values. Even if optional values were allowed, the program would have to ask questions for them anyway, since the user has no opportunity to specify them subsequent to the original command. We want the interaction to be more user-directed. Our general principle is to allow the user to volunteer additional information during the course of the interaction, as long as the path has not been determined and values remain unspecified. We use the following interaction protocol. The user enters the initial command and hits return. The program will accept additional lines of input. However, if the user just hits return, and the program needs more information, the program will generate a question. The user answers the question, followed by a return. As before, additional information may be entered on subsequent lines. If the user hits return on an empty line, another question is generated, if necessary.

Brodie[Br81] and Skuce[Sk80] both present systems for representing DB change. Skuce's goal is to provide an English-like syntax for DB procedure specification. Procedures have a rigid format and require all information to be entered at time of invocation in a specific order, as with any computer subprogram. Brodie is attempting to also specify DB procedures for DB change. He allows some information to be specified later, but the order is fixed. Neither allow the user to choose the order of entry, and neither accomodates

variants that would require different sets of values to be specified. However, like our method, and unlike Kaplan and Davidson[KD81], they attempt to model DB changes that correspond to real world actions rather than just specifying English synonyms for single DB commands.

Certain constraints on updates are implicit on verbgraphs, such as APPT.where ← input from R_3, which constrains the location of the meeting to be the office of one of the two employees. We also use verbgraphs to maintain database consistency. Integrity constraints take two forms: constraints on a single state and constraints on successive database states. The second kind is harder to enforce; few systems support constraints on successive states.

Verbgraphs provide many opportunities for specifying various defaults. First, we can specify default values, which may depend on other values. Second, we can specify default paths. Verbgraphs are also a means for specifying non-DB operations. For example, if an appointment is made with someone outside the company, generate a confirmation letter to be sent.

All of the above discussion has assumed we are selecting a variant where the sense has already been determined. In general sense selection, being equivalent to the frame selection problem in Artifical Intelligence[CW76], is very difficult. We do feel that verbgraph will aid in sense selection, but will not be as efficacious as for variant selection. In such a situation, perhaps the English parser can help disambiguate or we may want to ask an appropriate question to select the correct sense, or as a last resort, provide menu selection.

IV. AN ALTERNATIVE TO VERBGRAPHS

We are currently considering hierarchically structured transactions, as used in the TAXIS semantic model [MB80], as an alternative to verbgraphs. Verbgraphs can be ambiguous, and do not lend themselves to top-down design. Hierarchical transactions would seem to overcome both problems. Hierarchical transactions in TAXIS are not quite as versatile as verbgraphs in representing variants. The hierarchy is induced by hierarchies on the entity classes involved. Variants based on the relationship among particular entities, as recorded in the database, cannot be represented. For example, in the SCHEDULE-APPOINTMENT action, we may want to require that if a supervisor schedules a meeting with an employee not under his supervision, a message must be sent to that employee's supervisor. This variant cannot be distinguished by classifying one entity as a supervisor and the other as an employee because the variant does not apply when the supervisor is scheduling a meeting with his own employee. Also all variants in a TAXIS transaction hierarchy must involve the same entity classes, where we may want to involve some classes only in certain variants. For example, a variant of SCHEDULE-APPOINTMENT may require that a secretary be present to take notes, introducing an entity into that variant that is not present elsewhere.

We are currently trying to extend the TAXIS model so it can represent such variants. Our extensions include introducing guards to distinguish specializations and adding optional actions and entities to transactions. A guard is a boolean expression involving the entities and the database that, when satisfied, indicates the associated specialization applies. For example, the guard

scheduler in class(supervisor) and
scheduler ≠ supervisor-of(schedulee)

would distinguish the variant described above where an employee's supervisor must be notified of any meeting with another supervisor. The discrimination mechanism in TAXIS is a limited form of guards that only allows testing for entities in classes.

V. REFERENCES

[Br78] Brodie, M.L., Specification and verification of data base semantic integrity. CSRG Report 91, Univ. of Toronto, April 1978.

[Br81] Brodie, M.L., On modelling behavioral semantics of database. VLDB 7, Cannes France, Sept. 1981.

[CH81] Carbonell, J. and Hayes, P., Multi-strategy construction-specification parsing for flexible database query and update. CMU Internal Report, July 1981.

[CW76] Charniak, E. and Wilks, Y., Computation Semantics. North Holland, 1976.

[Ch76] Chen, P.P.-S., The entity-relationship model: toward a unified view of data. ACM TODS 1:1, March 1976, pp. 9-36.

[Co79] Codd, E.F., Extending the database relational model to capture more meaning. ACM TODS 4:4, December 1979, pp. 397-434.

[Da78] Damereau, F.J., The derivation of answers from logical forms in a question answering system. American Journal of Computational Linguistics. Microfiche 75, 1978, pp. 3-42.

[HM81] Hammer, M. and McLeod, D., Database description with SDM: A semantic database model. ACM TODS 6:3, Sept. 1981, pp. 351-386.

[Ha77] Harris, L.R., Using the database itself as a semantic component to aid the parsing of natural language database queries. Dartmouth College Mathematics Dept. TR 77-2, 1977.

[Ka79] Kaplan, S.J., Cooperative responses from a natural language data base query system. Stanford Univ. Heuristic Programming Project paper HPP-79-19.

[KD81] Kaplan, S.J., and Davidson, J., Interpreting Natural Language Updates. Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, June 1981.

[MB80] Mylopoulos, J., Bernstein, P.A., and Wong, H.K.T., A language facility for designing database-intensive applications. ACM TODS 5:2, June 1980, pp. 397-434.

[Sa78] Salveter, S.C., Inferring conceptual structures from pictorial input data. University of Wisconsin, Computer Science Dept., TR 328, 1978.

[Sa79] Salveter, S.C., Inferring conceptual graphs. Cognitive Science,3, pp. 141-166.

[Sk80] Skuce, D.R., Bridging the gap between natural and computer language. Proc. of Int'l Congress on Applied Systems, and Cybernetics, Acapulco, December 1980.

[Wa78] Walker, D.E., Understanding Spoken Language. American Elsevier, 1978.

[Wi81] Wiederhold, G., Kaplan, S.J., and Sagalowicz, D., Research in knowledge base management systems. SIGMOD Record, VII, #3, April 1981, pp. 26-54.

[Wo76] Woods, W., et. al., Speech Understanding Systems: Final Technical Progress Report. BBN No. 3438, Cambridge, MA, 1976.

[Wz75] Waltz, D., Natural language access to a large database: an engineering approach. In Proc. of the Fourth Int'l Joint Conf. on Artificial Intelligence, 1976.