Construct Algebra: Analytical Dialog Management

Alicia Abella and Allen L. Gorin
AT&T Labs Research
180 Park Ave. Bldg 103 Florham Park, NJ 07932

Abstract

In this paper we describe a systematic approach for creating a dialog management system based on a Construct Algebra, a collection of relations and operations on a task representation. These relations and operations are analytical components for building higher level abstractions called dialog motivators. The dialog manager, consisting of a collection of dialog motivators, is entirely built using the Construct Algebra.

1 INTRODUCTION

The dialog manager described in this paper implements a novel approach to the problem of dialog management. There are three major contributions: the task knowledge representation, a Construct Algebra and a collection of dialog motivators. The task knowledge representation exploits object-oriented paradigms. The dialog motivators provide the dialog manager with the dialog strategies that govern its behavior. The Construct Algebra provides the building blocks needed to create new dialog motivators and analyze them.

The first main component of this dialog manager is the task knowledge representation. The task knowledge is encoded in objects. These objects form an inheritance hierarchy that defines the relationships that exists among these objects. The dialog manager exploits this inheritance hierarchy in determining what queries to pose to the user. No explicit states and transitions need to be defined using this framework (Bennacef et al., 1996; Meng and et. al., 1996; Sadek et al., 1996). A change to the dialog does not require a change to the dialog manager, but more simply, a change to the inheritance hierarchy.

The second main component of this dialog manager is the collection of dialog motivators. The dialog motivators determine what actions need to be taken (e.g. ask a confirmation question). The dialog motivators are founded on a theoretical framework called a Construct Algebra. The Construct Algebra allows a designer to add new motivators in a principled way. Creating a new application requires defining the inheritance hierarchy and perhaps additional dialog motivators not encompassed in the existing collection.

This dialog manager has been used for two applications. The first is a spoken dialog system that enables a user to respond to the open-ended prompt *How may I help you?* (HMIHY) (Gorin et al., 1997). The system recognizes the words the customer has said (Riccardi and Bangalore, 1998) and extracts the meaning of these words (Wright et al., 1998) to determine what service they want, conducting a dialog (Abella and Gorin, 1997; Abella et al., 1996) to effectively engage the customer in a conversation that will result in providing the service they requested. The second application is to Voice Post Query (VPQ) (Buntschuh et al., 1998) which provides spoken access to the information in large personnel database (> 120,000 entries). A user can ask for employee information such as phone number, fax number, work location, or ask to call an employee. These applications are significantly different but they both use the same dialog manager.

2 Task Representation

Information about the task is defined using an object inheritance hierarchy. The inheritance hierarchy defines the relationships that exist amongst the task knowledge. Objects are defined to encode the hierarchy. This representation adheres to the principles of object-oriented design as described in (Booch, 1994). Each of the objects has three partitions. The first partition contains the name of the object, the second contains a list of variables with associated values that are specific to the object, and the third partition contains any methods associated with the object. For simplicity of illustration we will not include any of the methods. Each of the objects inherits its methods from a higher level object called the Construct. The Construct's methods are the relations and operations that will be described in section 4.

The result of the speech recognizer is sent to the spoken language understanding (SLU) module. The SLU module extracts the meaning of the user's utterance and produces a list of possible objects with associated confidence scores that is interpreted by the dialog manager. The dialog manager then uses the inheritance hierarchy and an algorithm[1] fully described in (Abella and Gorin, 1997) to produce a set of semantically consistent inputs to be used by the dialog manager. The input is represented as a boolean expression of constructs extracted from the utterance. This input is then manipulated by the dialog motivators to produce an appropriate action, which most often consists of playing a prompt to the user or generating a query to a database.

3 The Construct

A construct is the dialog manager's general knowledge representation vehicle. The task

[1]An understanding of this algorithm is not necessary for the understanding of the work described in this paper.

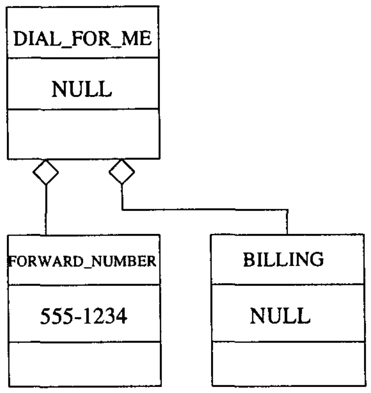

Figure 1: A construct example for HMIHY

knowledge is encoded as a hierarchy of constructs. The construct itself is represented as a tree structure which allows for the building of a containment hierarchy. It consists of two parts, a *head* and a *body*. Figure 1 illustrates a construct example for HMIHY. The DIAL_FOR_ME construct is the *head* and it has two constructs for its *body*, FORWARD_NUMBER and BILLING. These two constructs represent the two pieces of information necessary to complete a call. If a user calls requesting to place a call it is the DIAL_FOR_ME construct that is created with the generic BILLING construct and the FORWARD_NUMBER construct with its value set to empty. The dialog manager will then ask for the forward number and for the type of billing method. In figure 1 the dialog manager has received a response to the forward number request.

4 Construct Algebra

The construct algebra defines a collection of elementary relations and operations on a set of constructs. These relations and operations are then used to build the larger processing units that we call the dialog motivators. The set of dialog motivators defines the application. In this section we formally define these relations and operations.

4.1 The Construct

Definition 1 Head
A *head* is an ordered pair <*name,value*>, where *name* belongs to some set of prede-

fined names, N, and *value* belongs to some set of predefined values, V. A value may be NULL (not assigned a value).

Definition 2 Construct
A *construct* is defined recursively as an ordered pair <*head,body*> where *body* is a (possibly empty) set of constructs.

4.2 Relations

The Construct Algebra defines six relations in the set of constructs. In each of the definitions, c_1 and c_2 are constructs. Note that the symbols $\subseteq$ and $\subset$, introduced here, should not be understood in their usual "subset" and "proper subset" interpretation but will be described in definitions 4 and 5.

Definition 3 Equality
Two constructs are equal, denoted $c_1 = c_2$ when

$$head(c_1) = head(c_2) \text{ and}$$
$$body(c_1) = \text{body}(c_2)$$

Definition 3 requires that the heads of c_1 and c_2 be equal. Recall that the head of a construct is an ordered pair <*name,value*> which means that their names and values must be equal. A value may be empty (NULL) and by definition be equal to any other value. The equality of bodies means that a bijective mapping exists from the body of c_1 into the body of c_2 such that elements associated with this mapping are equal.

Definition 4 Restriction
c_1 is a restriction of c_2, denoted $c_1 \subseteq c_2$, when

$head(c_1) = head(c_2)$ and
$(\exists f : body(c_1) \to body(c_2))(f \text{ is 1 to 1} \wedge (\forall b_1 \in body(c_1))(b_1 \subseteq f(b_1))$

Intuitively, c_1 can be obtained by "pruning" elements of c_2. The second part of the definition, $(\exists f : \ldots)$ is what differentiates $\subseteq$ from $=$. It is required that a mapping f between the bodies of c_1 and c_2 exist with the following properties:

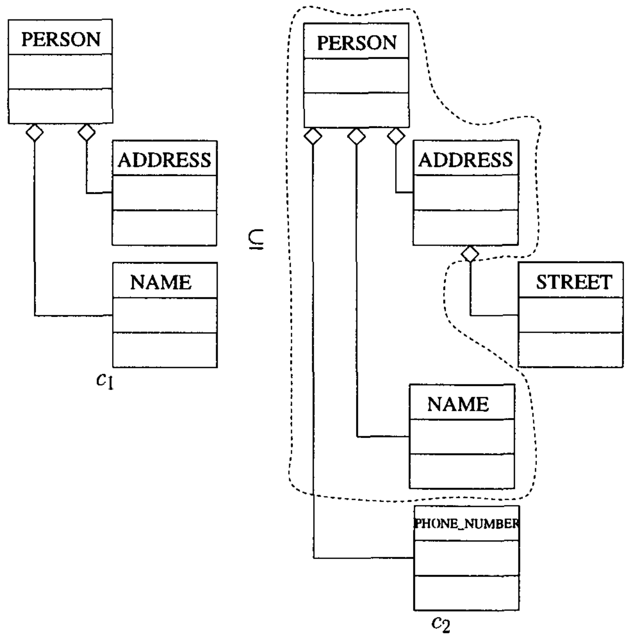

Figure 2: STREET and PHONE_NUMBER are "pruned" from c_2 to obtain c_1.

- f is 1 to 1. In other words, different elements of the body of c_1, call them b_1, are associated with different elements of the body of c_2, call them b_2
- The elements of the body of c_1 are restrictions of the elements of the body of c_2. In other words, $b_1 \subseteq b_2$, where b_1 are elements from the body of c_1 and b_2 are elements from the body of c_2.

Figure 2 illustrates an example.

Definition 5 Containment
c_1 is contained in c_2, denoted $c_1 \subset c_2$, when

$c_1 \subseteq c_2$ or $(\exists b_2 \in body(c_2))(c_1 \subset b_2)$

We assume that $c_1 \subset c_2$ either if c_1 is a restriction of c_2 or if c_1 is contained in any element of the body of c_2. Figure 3 gives an example. The AMBIGUITY construct represents the fact that the system is not sure whether the user has requested a COLLECT call or a CALLING_CARD call. This would trigger a clarifying question from the dialog manager.

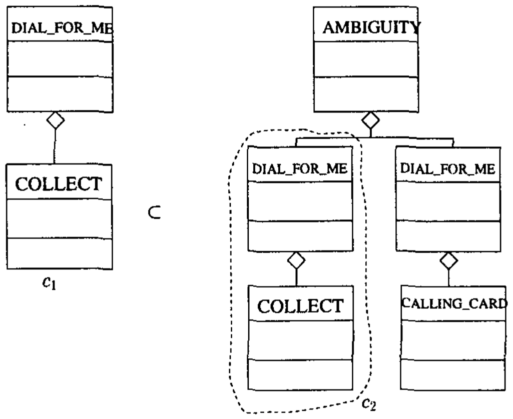

Figure 3: $c_1 \subset c_2$

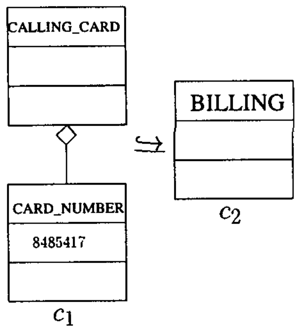

Figure 4: $c_1 \underline{\hookrightarrow} c_2$

Definition 6 Generalization
c_2 is a generalization of c_1, denoted $c_1 \underline{\hookrightarrow} c_2$, when

$$head(c_1) \underline{\hookrightarrow} head(c_2) \text{ and}$$
$$(\exists f : body(c_2) \rightarrow body(c_1))$$
$$(f \text{is 1 to 1} \wedge (\forall b_2 \in body(c_2)))(f(b_2) \underline{\hookrightarrow} b_2)$$

The generalization of heads means that the *name* of c_2 is on the inheritance path of c_1 and their values are equal. Intuitively, c_2 is an ancestor of c_1 or in object-oriented terms "c_1 is-a c_2". Note the similarity of this relation to $\subseteq$. Figure 4 illustrates an example. BILLING is a generalization of CALLING_CARD, or in other words CALLING_CARD is-a BILLING.

Definition 7 Symmetric Generalization
c_1 is a symmetric generalization of c_2, denoted $c_1 \simeq c_2$, when

$c_1 \underline{\hookrightarrow} c_2$ or $c_2 \underline{\hookrightarrow} c_1$

This definition simply removes the directionality of $\underline{\hookrightarrow}$. In other words, either "c_1 is-a c_2" or "c_2 is-a c_1".

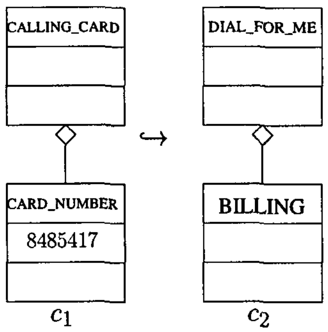

Figure 5: $c_1 \hookrightarrow c_2$

Definition 8 Containment Generalization
c_1 is a containment generalization of c_2, denoted $c_1 \hookrightarrow c_2$, when

$(\exists b_2)(b_2 \subset c_2 \wedge b_2 \simeq c_1)$

b_2 is contained in c_2 and c_1 is a symmetric generalization of b_2. An example is illustrated in figure 5. BILLING is contained in DIAL_FOR_ME and is a symmetric generalization of CALLING_CARD.

4.3 Operations

The Construct Algebra consists of two operations union, $\cup$ and projection, $\backslash$.

Definition 9 Union ($\cup$)
We will define this operation in several steps. Each step is a progression towards a more general definition.

Definition 9.1 Union of values ($v_1 \cup v_2$)

$$v_1 \cup v_2 = \begin{cases} v_1, & v_1 = v_2 \text{ and } v_1 \neq \text{NULL} \\ v_2, & v_1 = v_2 \text{ and } v_1 = \text{NULL} \\ \text{not defined}, & v_1 \neq v_2 \end{cases}$$

Recall that by definition, NULL is equal to any other value.

Definition 9.2 Union of heads

We define $head(c_1) \cup head(c_2)$ only in the case $c_1 \underline{\hookrightarrow} c_2$, which is all that is needed for a definition of $\cup$.

$$head(c_1) \cup head(c_2) = (name(c_1), value(c_1) \cup value(c_2))$$

Definition 9.3 ($c_1 \cup c_2$)

If $c_1 \underline{\hookrightarrow} c_2$,

$$\begin{aligned} c_1 \cup c_2 = \\ &(head(c_1) \cup head(c_2), \\ &\{f(b_2) \cup b_2 | b_2 \in body(c_2)\} \cup \\ &\{b_1 | b_1 \in body(c_1) \wedge \\ &(\forall b_2 \in body(c_2))(b_1 \neq f(b_2))\}) \end{aligned}$$

In this definition the head of the resulting construct is the union of the heads of the operands. The body of the resulting construct consists of two parts. The first part is a set of unions (denoted $f(b_2) \cup b_2$ in the definition above) where b_2 spans the body of the second operand c_2 and f is a mapping from Definition 6. Recall that the mapping f associates elements of the $body(c_1)$ with elements of the $body(c_2)$ such that $f(b_2) \underline{\hookrightarrow} b_2$ for $b_2 \in body(c_2)$ so the union $f(b_2) \cup b_2$ is (recursively) defined in Definition 9.3. The second part of the body of the resulting construct consists of those elements b_1 of the $body(c_1)$ that no element from the $body(c_2)$ maps into through the mapping f. In other words, the second part of the body consists of those elements "left behind" in the $body(c_1)$ after the mapping f. Figure 6 illustrates an example. The union operations results in a construct with the head CALLING_CARD and a body that contains both CARD_NUMBER and EXPIRATION. The CARD_NUMBER construct from c_1 and c_2 can be combined because the value of CARD_NUMBER from c_1 is NULL. The construct EXPIRATION is added because it does not exist on the body of c_2.

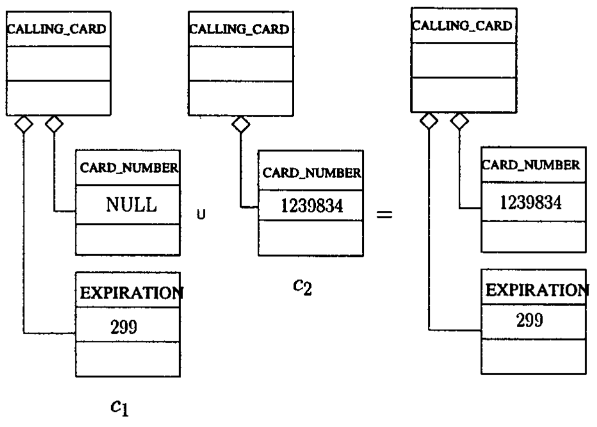

Figure 6: $c_1 \cup c_2$ if $c_1 \underline{\hookrightarrow} c_2$

Definition 9.4 $c_1 \cup c_2$

If $c_1 \simeq c_2$,

$$c_1 \cup c_2 = \begin{cases} c_1 \cup c_2, & c_1 \underline{\hookrightarrow} c_2 \\ c_2 \cup c_1, & c_2 \underline{\hookrightarrow} c_1 \end{cases}$$

Definition 9.5 $c_1 \cup c_2$

If $c_1 \hookrightarrow c_2$,

$$c_1 \cup c_2 = \begin{cases} c_1 \cup c_2, & c_1 \simeq c_2 \\ (head(c_2), & \\ \{c_1 \cup b_2 | b_2 \in body(c_2) \wedge c_1 \hookrightarrow b_2\} \cup & \\ \{b_2 | b_2 \in body(c_2) \wedge c_1 \not\hookrightarrow b_2\}), & c_1 \not\simeq c_2 \end{cases}$$

Figure 7 illustrates this union. The head of the resulting construct is the head of c_2 which is DIAL_FOR_ME. The resulting construct no longer has BILLING but

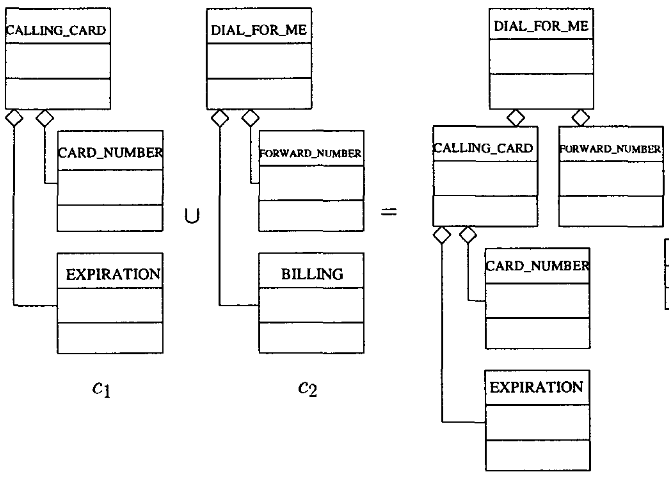

Figure 7: $c_1 \cup c_2$ if $c_1 \hookrightarrow c_2$

rather CALLING_CARD since BILLING is a generalization of CALLING_CARD. In addition the resulting construct contains the construct FORWARD_NUMBER because it remains from DIAL_FOR_ME.

Definition 9.6 $c_1 \cup c_2$
In the general case,
$$c_1 \cup c_2 = \begin{cases} c_1 \cup c_2, & c_1 \hookrightarrow c_2 \\ c_2 \cup c_1, & c_2 \hookrightarrow c_1 \\ ((\text{REP}, \text{NULL}), \{c_1, c_2\}), & c_1 \not\hookrightarrow c_2 \text{ and } \\ & c_2 \not\hookrightarrow c_1 \end{cases}$$

In this definition REP is a construct used to represent the union of those constructs that do not satisfy any of the aforementioned conditions. By definition REP has a value of NULL and the body consists of the constructs c_1 and c_2.

Definition 10 Projection (\)

$$c_1 \backslash c_2 = \begin{cases} ((\text{AMBIGUITY}, \text{NULL}), & \\ \{b_1 \cup c_2 | b_1 \subset c_1 \wedge b_1 \simeq c_2\}) & c_2 \hookrightarrow c_1 \\ c_1 & c_2 \not\hookrightarrow c_1 \end{cases}$$

Figure 8 illustrates an example of an ambiguous construct and the result of the

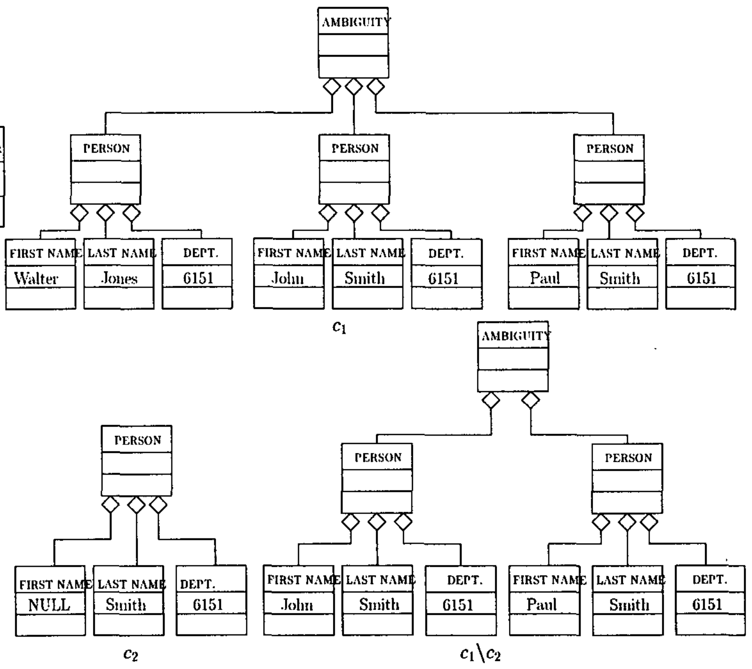

Figure 8: Projection operation example

projection operation. The construct is AMBIGUITY because all the elements of its *body* have the value of 6151 for DEPT. In this example, c_2 contains the construct LAST_NAME with the value of Smith. There are 2 constructs on the body of c_1 that are in the relation $b_2 \subset c_1$, in other words have value for LAST_NAME of Smith. Therefore the result is an AMBIGUITY construct with two elements on its body, both with the LAST_NAME value of Smith.

5 Dialog Motivators

A dialog motivator determines what action the dialog manager needs to take in conducting its dialog with a user. The dialog manager for HMIHY currently consists of 5 dialog motivators. They are *disambiguation* , *confirmation, error handling* (recovery from misrecognition or misunderstanding and silence), *missing information* and *context switching.* VPQ uses two additional motivators, they are *continuation* and

c_Q: Construct used for disambiguation, $c_Q \in c$
c_A: User response

$$D_k(c, c_{IDK}) = \begin{cases} c, & c \not\subseteq \text{AMBIGUITY} \\ D_{k+1}(c, c_{IDK}), & c_A \underline{\hookrightarrow} \text{ERROR} \\ D_{k+1}(c, c_{IDK} \cup c_Q), & c_A = \text{IDK} \\ c \backslash c_A, & c_A \hookrightarrow c \\ c_A & c_A \not\hookrightarrow c \end{cases}$$

Figure 9: Disambiguation Motivator

database querying.

The *disambiguation* motivator determines when there is ambiguous semantic information, like conflicting billing methods. *Confirmation* is used when the SLU returns a result with low confidence. *Error handling* takes on three forms. There is error recovery when the speech recognizer has likely misrecognized what the user has said (low confidence scores associated with the recognition results), when the user falls silent, and when the user says something the SLU does not expect or does not handle. *Missing information* determines what information to ask about in order to complete a transaction. *Context switching* is the ability of the system to realize when the user has changed his/her mind or realizes that it has misunderstood and allows the user to correct it. The *continuation* motivator determines when it is valid to offer the user the choice to query the system for additional information. *Database querying* decides when the system has acquired enough information to query a database for the requested information.

5.1 Disambiguation Motivator

Figure 9 illustrate how the disambiguation motivator is created using the Construct Algebra. The disambiguation motivator is called with the current construct c and a set of constructs called c_{IDK} that represents information that the user does not know (IDK - "I Don't Know"), in other words, the user explicitly responds to a prompt with the phrase "I don't know" or its equivalent[2].

[2]The phrases chosen are based on trials

Input: A sequence of semantic input from the SLU module in response to a prompt
Output: Complete construct c (no need for further dialog)

```
Repeat
    For all dialog motivators DM_i
        if DM_i applies to $c$
            Perform action(DM_i,$c$)
            Apply Dialog Manager to get $c_A$
            Using Construct Algebra,
            combine $c$ and $c_A$ into $c$
Until no motivator applies
Return $c$
```

Figure 10: Dialog Manager algorithm

The motivator runs through several checks on the construct c. The first is to check to see if in fact the motivator applies, or in other words if c is a restriction of AMBIGUITY. If it is not then the motivator simply return c without changing it. The second step is to check to see if the ERROR construct is a generalization of c_A where c_A represents the user's response. The ERROR construct represents an error condition like silence or misrecognition. If it is, then it goes on to next motivator because this motivator does not apply to error conditions. If c_A equals the IDK construct then this means that the user did not know the answer to our query and we add the construct used for disambiguation, c_Q to the set of constructs c_{IDK}. If however, c_A is in the containment generalization relation with c then the projection operation is applied and the result is returned. If c_A is not in this relation then this indicates a context switch on the part of the user and the disambiguation motivator returns c_A as the result.

All other motivators are constructed in a similar fashion. An application can use these motivators or create new ones that are application specific using the operations and relations of the Construct Algebra.

System: VPQ. What can I do for you?
User: I need the phone number for Klein.
System: I have more than 20 listings for Klein. Can you please say the first name?
User: William.
System: I have 2 listings for William Klein. Can you tell me the person's work location?
User: Bedminster
System: The phone number for William Klein is 973 345 5432. Would you like more information?
User: No.
System: Thank you for using VPQ.

Figure 11: A sample dialog for VPQ

6 Dialog Manager

The input to the dialog manager is a collection of semantic input generated by the SLU. Figure 10 illustrates the algorithm used by the dialog manager. The output is the complete construct c which no longer requires further dialog. The algorithm loops through all the dialog motivators determining which one needs to be applied to c. If it finds a motivator that applies then it will perform the necessary action (e.g. play a prompt or do a database lookup). The algorithm repeats itself to obtain c_A (the construct answer). In other words, the construct that results from the action is subject to the dialog motivators starting from the beginning. Once c_A has been found to be complete it is combined with c using Construct Algebra to produce a new construct. This new construct c also goes through the loop of dialog motivators and the procedure continues until no motivator applies and the algorithm returns the final construct c.

6.1 Example

To illustrate how the dialog manager functions we will use an example from VPQ. Figure 11 illustrates a sample dialog with the system. The sequence of motivators for VPQ is *error handling*, *confirmation*, *missing information*, *database querying* and *disambiguation*. The construct that is created as a result of the user's initial utterance is shown in figure 12. All the information needed to do a database lookup is found in the user's utterance, namely the piece of information the user is seeking and the name of the person. Therefore the first motivator that applies is *database querying*. This motivator creates the database query and based on the result creates the construct c_A. The construct c_A is then searched by each of the motivators beginning again with *error handling*. The motivator that applies to c_A is the disambiguation motivator because there are more than 20 people in the database whose last name is pronounced Klein, including Klein, Cline and Kline. The disambiguation motivator searches through c_A to determine, based on preset parameters, which piece of information is most useful for the disambiguation process as well as which piece of information the user is likely to know, which is selected when the inheritance hierarchy is designed. For VPQ this includes asking about the first name and work location. In this example the dialog manager searches the database entries and determines that the most discriminating piece of information is the first name. Once the user responds with the first name there are still 2 possible candidates and it asks for the next piece of information which is work location. Had the user not known the work location the system would have read out the phone number of both people since the total number of matches is less than 3. If the number of entries after disambiguation remains greater than 3 the system refers the user to a live operator during work hours.

7 Conclusion

In this paper we have described a novel approach to dialog management. The task knowledge representation defined intuitively and without the need to define call flows in the traditional finite-state approach. The Construct Algebra serves as the building blocks from which the dialog motivators that drive the dialog system are comprised. Building a new application will only require the designer to define the objects (e.g. COL-

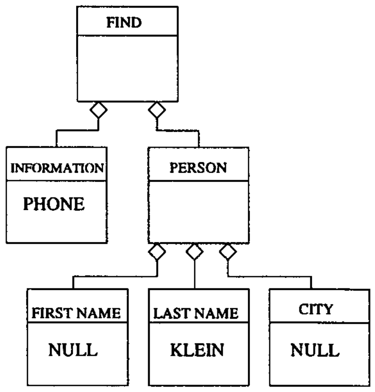

Figure 12: Sample construct for VPQ.

LECT, CREDIT etc.) and the inheritance hierarchy. The Construct Algebra serves as an analytical tool that allows the dialog motivators to be formally defined and analyzed and provides an abstraction hierarchy that hides the low-level details of the implementation and pieces together the dialog motivators. This same dialog manager is currently being used by two very different applications (HMIHY and VPQ).

References

Alicia Abella and Allen L. Gorin. 1997. Generating semantically consistent inputs to a dialog manager. In *Proc. EuroSpeech Rhodes, Greece.*

A. Abella, M. K. Brown, and B. Buntschuh. 1996. Development principles for dialog-based interfaces. *European Conference on Artificial Intelligence.*

S. Bennacef, L. Devillers, S. Rosset, and L. Lamel. 1996. Dialog in the railtel telephone-based system. *International Conference on Spoken Language Processing.*

Grady Booch. 1994. *Object-Oriented Analysis and Design with Applications.* Benjamin Cummings.

B. Buntschuh, C. Kamm, G. DiFabbrizio, A. Abella, M. Mohri, S. Narayan, I. Zeljvokic, R.D. Sharp, J. Wright, S. Marcus, J. Shaffer, R. Duncan, and J.G. Wilpon. 1998. VPQ: A spoken language interface to large scale directory information. In *Proc. ICSLP Sydney.*

A.L. Gorin, G. Riccardi, and J.H. Wright. 1997. How May I Help You? *Speech Communciation.*

Helen Meng and Senis Busayapongchai et. al. 1996. Wheels: A conversational system in the automobile classifieds domain. *International Conference on Spoken Language Processing.*

G. Riccardi and S. Bangalore. 1998. Automatic acquisision of phrase grammars for stochastic language modeling. In *Proc. ACL Workshop on Very Large Corpora, Montreal.*

M.D. Sadek, A. Ferrieux, A. Cozannet, P. Bretier, F. Panaget, and J. Simonin. 1996. Effective Human-Computer Cooperative Spoken Dialogue: the AGS Demonstrator. *International Conference on Spoken Language Processing.*

Jerry Wright, Allen L. Gorin, and Alicia Abella. 1998. Spoken language understanding within dialogs using a graphical model of task structure. In *Proc. ICSLP Sydney.*