ABSTRACT

Consider a conversational speech system incorporating spoken language understanding (SLU) and dialog manager (DM) modules. A prerequisite of natural dialog is that understanding takes place in context, which in turn necessitates a sharing of task knowledge between the two modules. However, because they are engaged in different activities it is often the case that the task knowledge representation is different for each of them, an undesirable duplication. In this paper we consider AT&T’s How May I Help You?SM natural dialog system. We present a method for automatically inducing the task knowledge representation used for spoken language understanding from that used for dialog management. Also, the context information needed by the SLU module for understanding each utterance is generated automatically by the DM from its own knowledge representation and the current dialog situation. This enables understanding-in-context to be implemented while avoiding the duplication involved in creating and maintaining separate task knowledge representations. The system has been evaluated using a database comprising 200k dialogs drawn from live customer traffic.

1. INTRODUCTION

Spoken dialog systems typically contain spoken language understanding (SLU) and dialog manager (DM) modules, each of which uses a representation of the task knowledge. The SLU needs this for contextual understanding: interpreting an utterance in the context of the ongoing dialog [1]. Performance in this regard is highly correlated with user satisfaction [2]. Furthermore, if the speech recognizer can provide a word lattice or graph then use of dialog expectation (derived from context) can improve understanding performance [3]. In this paper we consider AT&T’s How May I Help You?SM natural dialog system [4]. The task knowledge representation used by the DM is an object-oriented inheritance hierarchy [5,6]. A similar approach was subsequently used in [7]. This is a very efficient structure for encoding relationships among the task classes, and enables the call flow for each application to be implemented independently of the core DM algorithms. The knowledge representation used by the SLU for contextual understanding is a graph structure which is again expressed as application-specific data [8]. Here we present a method for automatically inducing the latter representation from the former, removing the need to maintain separate versions. The DM must also send the context information to the SLU with each user utterance, and this information is generated automatically from the inheritance hierarchy and the current dialog situation.

In section 2 we consider the task knowledge and context issues in more detail. Section 3 contains a description of how the SLU module uses this information, including a new semantic parser for finding alternative surface readings of an utterance. Some results for a recent Customer Care application are presented in section 4.

2. TASK KNOWLEDGE REPRESENTATION

In this section we consider the two forms of task knowledge representation used by the DM and SLU modules.

2.1 DM Inheritance Hierarchy

For dialog management, information about the task is defined using a set of classes together with relationships between them, forming an object inheritance hierarchy [5,6]. The classes are of three types: call-types, named-entities, and nonterminals. The call-types are machine actions or services that a user may request. The named-entities are attributes that require a value in order to implement a service, such as a phone number or a dollar amount. The nonterminals are useful for grouping classes or invoking clarification prompts. The relationships between the classes are of two kinds. The inheritance or is-a relation can connect two classes of similar type. This expresses the fact that a more specific class inherits the properties of a more general class. The containment or has-a relation can connect any two classes, and expresses the fact that a class (usually a call-type) has certain required items of information for its realization.

Using the inheritance hierarchy in conjunction with a task-independent set of dialog motivators [6], the DM is able to conduct a dialog with the user leading to completion of the service or services requested, with no need to explicitly define states and transitions. The dialog call flow is therefore an emergent property of the inheritance hierarchy. For example, the Missing-Information motivator is triggered by the has-a relation to request some information from the user. At any time the actual situation within a dialog is represented as a construct: a tree constructed from the inheritance hierarchy using the containment and inheritance relations. At each turn in the dialog the result of the understanding module is transformed into another construct, and the dialog motivators map this pair of constructs to a new one representing the updated situation.

2.2 SLU Task Structure Graph

The understanding module exploits the task knowledge in the
form of a task structure graph [8]. This differs from the inheritance hierarchy in being designed to support understanding-in-context. In particular, when an utterance can only be disambiguated with respect to a particular context, this is reflected in the task structure graph by a node (representing a class) from which several arcs emanate. There are two aspects to the graph: the structure itself, in the form of the arcs that connect the classes, and the sets of classes that are to be considered “exclusive” for semantic parsing (section 3.2).

Structurally the graph is similar to the inheritance hierarchy. The set of classes is the same, with the exception of a base “Concept” class used only by the DM. Let the containment relation be represented by \( Y \circ X \) (\( Y \) has-a \( X \)), and the inheritance by \( Y \leftarrow X \) (\( X \) is-a \( Y \)). The arcs are created in four steps:

1. For each containment relation of the form \( Y \circ X \), set arc \( \overrightarrow{XY} \).
2. For each call-type or nonterminal class \( X \) with no containment (no \( Z \) such that \( Z \circ X \)) where \( Y \leftarrow X \), set arc \( \overrightarrow{YX} \). Furthermore, if for a given \( Y \) the set of all such \( X \) contains more than one element then this forms an exclusive set.
3. For each named-entity or nonterminal class \( X \) that is contained in another class (\( \exists Z: Z \circ X \)), for each other named-entity class \( Y \) such that \( X \leftarrow Y \), set arc \( \overrightarrow{YX} \), and continue recursively: set arc \( \overrightarrow{UY} \) for each \( U \) such that \( Y \leftarrow U \), and so on.
4. For each named-entity class \( X \) not captured in steps 1 or 3, for each other named-entity class \( Y \) such that \( X \leftarrow Y \) add arc \( \overrightarrow{XY} \); note that these arcs reverse the is-a relation.

If the inheritance hierarchy is well-formed in that every class is either contained in or inherits from a class that is connected (via similar relations) to the base class, then the graph constructed in this way is similarly well-formed. Step 4 covers the situation where a particular named-entity class may be used for several purposes in an application, e.g. a phone number may be a number to call, a number to bill a call to, or a number already called. In the graph for the Operator Services application (section 4.1), the node representing the class Phone_Number has three arcs to Forward_Number, Billed_Number and Dialed_Number respectively, all of which inherit from Phone_Number in the inheritance hierarchy. When the user is asked for a phone number and provides it, the dialog context makes clear which purpose is intended.

The exclusive sets of classes generated in step 2 guide the semantic parser. Generally these represent mutually exclusive alternatives, but it is easy to intervene manually if such classes are permitted to co-occur.

2.3 Context of an Utterance

In any conversational system, understanding requires a context. The minimum possible context would be the single class that is the focus for the latest turn in the dialog, but this is insufficient because the user’s reply may have significance for a broader context. On the other hand, it is not necessary to take the entire dialog history as the context because there are often equivalent possible histories that lead to the same situation conceptually.

The construct (section 2.1) representing the current situation within the dialog is treated here as the pertinent context for understanding. This is a tree structure formed using the inheritance hierarchy containment and inheritance relations. At each turn, the focus class is a leaf of this tree. The context information sent to the SLU is generated automatically by the DM, and consists of the path from this class to the root of the tree, with the inheritance information interleaved with the containment. E.g. the context string

\[
X_0 \rightarrow X_4 \rightarrow X_1 \rightarrow X_6 \rightarrow X_5 \rightarrow X_2 \rightarrow X_7 \rightarrow X_3
\]

(where each \( X_i \) is a class and \( X_0 \) is the root) can represent the following situation:

\[
X_0 \circ X_4 \circ X_1 \circ X_6 \circ X_5 \circ X_2 \circ X_7
\]

The understanding module parses the context string and relates the latest user utterance to it using the task structure graph.

3. SPOKEN LANGUAGE UNDERSTANDING

The two most important components of the SLU module are the classifier and the interpreter.

3.1 SLU Classifier Component

For robust classification of utterances we use salient grammar fragments acquired automatically from a corpus of transcribed and labelled training data [4,9]. For the new semantic parser we associate with each grammar fragment a “positive significance set” of classes with which the fragment is significantly correlated, defined for fragment \( f_i \) and significance level \( \alpha \) as

\[
C_i^+ = \left\{ c_k : P(X_{ik} \geq x_{ik} \mid N, n_k, N_i) \leq \alpha \right\}
\]

where \( N \) is the number of training utterances, \( n_k \) is the number of these labelled with class \( c_k \), \( N_i \) is the number containing a match of fragment \( f_i \), and \( X_{ik} \) is the number satisfying both. The hypergeometric probability gives an exact significance test:

\[
P(X_{ik} = x_{ik} \mid N, n_k, N_i) = \binom{n_k}{x_{ik}} \binom{N-n_k}{N_i-x_{ik}} / \binom{N}{N_i}
\]

Given a lattice of grammar fragment matches for a test utterance, the union of these sets over the matched fragments is the overall positive significance of that utterance for the classes:

\[
C^+ = \bigcup_{f_i \text{ matched}} C_i^+
\]

From this we induce a set of semantic parses as follows:

1. \( R^{(0)} = C^+, \ t = 1 \).
2. Initialize \( S^{(t)} \) with some \( c_k \in R^{(t-1)} \).
3. Add to \( S^{(t)} \) all \( c_j \in C^+ \) that are compatible with \( c_k \).
4. \( R^{(t)} = R^{(t-1)} - S^{(t)} \).
5. If \( R^{(t)} \neq \emptyset \) then \( t = t + 1 \) and continue from 2.
The definition of “compatible” used in step 3 is based on the exclusive sets derived from the inheritance hierarchy (section 2.2): two classes are incompatible either if they are members of the same exclusive set, or if all the parent classes (within the task structure graph) of one are incompatible with the other. This recursively extends the direct definition of exclusiveness to classes that are incompatible by implication. Naturally, two classes are compatible if they are not incompatible.

Each semantic parse \(S^{(t)}\) then represents a subset of the classes and an associated subset of the fragment matches, and is maximal in that no further classes associated with any of the matches can be added without incompatibility. These are treated as meaningful combinations of events as defined by the task knowledge. Some parses may be caused by a speech recognition error or by an unorthodox use of a word or phrase. Others may represent distinct service requests. To arrive at a ranking of the parses, a score is first derived for each class using measured posterior distributions over the classes for the grammar fragments, obtained during training. The ranking is then based on a combination of criteria including these scores, coverage of the utterance by the matches within a parse, and consistency with dialog context of the classes. Further details of these steps will be published elsewhere. The output of the classifier is a set of classes with scores, representing a surface reading of the utterance.

### 3.2 SLU Interpreter Component

The purpose of the interpreter component is to follow the implications of the latest utterance for the dialog, taking account of the context. To do this, an inference is performed on the task structure graph (section 2.2) which uses the following information:

1. Class scores for the highest-ranked parses, from the classifier.
2. Scores for explicit affirmation or denial, if permitted by the context.
3. Context information sent from the DM (section 2.3).

The context information takes the form of a path into the graph from the root node. The other end of the path is the focus for the latest turn, and any explicit affirmation or denial refers to that focus. A three-pass algorithm propagates evidence through the graph. During the first pass, evidence supporting and opposing the focus class is gathered and then a score for the focus class is assigned, (implicit confirmation is permitted). During the second and third passes, scores are assigned for all the remaining classes. One guiding principle is to minimize the scope of any shift of context indicated by the evidence: any part of the context may be in error but it is preserved as far as possible. Further details of the inference are given in [8].

### 3.3 Understanding Representation

The understanding representation returned to the DM consists of a set of classes with scores, together with the values of any named-entities (such as phone numbers or dollar amounts) that the user has provided in the latest utterance. This informs the DM about the way in which the latest utterance extends, supports or contradicts the current context. This information is processed by the dialog motivators to initiate the next turn.

## 4. RESULTS

### 4.1 Applications

The understanding and dialog system described in this paper has been used in three applications: Operator Services [4], Directory Information [10], and most recently Customer Care [11]. In this application, customers are typically asking for information about bills or calling plans, or to change some aspect of their long-distance telephone service. The complexity of this application at all levels is higher than for the previous ones [11]. On the one hand, users often say something rather short but vague, e.g. “I have a query about something on my bill”, which requires further dialog for clarification. On the other hand, some utterances are long and complex and often require confirmation because of low confidence. The longer utterances are also prone to sporadic matches of grammar fragments which may be misleading, but the semantic parsing and ranking alleviates this: 8% of utterances have two semantic parses (section 3.1), and 1% have three or more. It is also necessary to obtain information about the customer, for example the telephone number for their account. Some examples of classes for the Customer Care application are as follows:

<table>
<thead>
<tr>
<th>Call-types</th>
<th>Named-entities</th>
<th>Nonterminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billing_Credit</td>
<td>Phone_Number</td>
<td>General_Billing</td>
</tr>
<tr>
<td>Account_Balance</td>
<td>Home_Number</td>
<td>Billing_Admin</td>
</tr>
<tr>
<td>Rate_Calling_Plans</td>
<td>Item_Amount</td>
<td>Caller</td>
</tr>
</tbody>
</table>

There are 28 call-types and 13 named-entities in total. Some relations between these classes are as follows:

General_Billing ← Billing_Credit
General_Billing ← Billing_Admin ← Account_Balance
Phone_Number ← Home_Number
Concept ← Caller
Billing_Credit ◁ Caller
Account_Balance ◁ Caller
Rate_Calling_Plans ◁ Caller
Caller ◁ Home_Number

One call-type (“Other”) is distinguished as a semantically open class to cover all cases not covered elsewhere, and these need to be transferred to a human representative.

### 4.2 Dialogs from Customer Care

The following are two examples of Customer Care dialogs:

1. **System**: AT&T How may I help you?  
   **User**: “I have a question on a bill”  
2. **System**: Okay. What is your question?  
   **User**: “I’m being charged ten cents a minute I’m under five cents a minute plan”  
3. **System**: I can help you with that. Are you calling from your home phone?  
   **User**: “No”  
4. **System**: May I have your home phone number starting with the area code?  
   **User**: “Nine zero eight *******”  
5. **System**: Was that nine zero eight *******?
User: “Yes” 6. System: I'm accessing your account

The customer’s request in turn 1 is vague but in turn 2 is very clear (the call-type is Rate_Calling_Plans) so confidence is high and no confirmation is required. It remains only to identify the caller.

1. System: AT&T How may I help you?
   User: “I have a I need to find out if we have a credit on our phone bill”
2. System: Do you need credit for a call on your bill?
   User: “I need to know what my bill payment is”
3. System: Would you like to hear your account balance?
   User: “Yes”
4. System: I can help you with that. Are you calling from your home phone?
   User: “I’m calling from my home”
5. System: I’m accessing your account

In turn 1 the phrase “have a credit” is misunderstood as a request for a billing credit but the confidence is low, so the next prompt is a confirmation which allows the user to correct the error. The broader context of a billing issue remains unchanged but the specific service request is updated (to Account_Balance). In turn 4 the user’s response is understood as an affirmation, specific to this context.

4.3 Performance on Utterances

For utterance-level evaluation we plot ROC curves of Correct Classification Rate (proportion correct among classified utterances) against False Rejection Rate (proportion rejected among those not labelled “Other”, section 4.1), for rank-1 and rank-2. Figure 1 shows ROC curves for a test set of utterances for the Customer Care application, drawn from a representative sample of dialog contexts. The dashed curves show what happens when the actual dialog context for each utterance is replaced with a generic context. The severe impact on performance is attributable to the resulting appearance of call-types or explicit confirmations which are inappropriate in many dialog contexts.

5. CONCLUSIONS

We have presented a method for automatically inducing SLU task structure from the object-oriented class hierarchy used by the DM, and from which the actual dialog call flow is an emergent property. Together with the automatic generation of context information by the DM for sending to the SLU accompanying each user utterance, this enables understanding-in-context to be implemented while avoiding the duplication of effort in creating and maintaining separate task knowledge representations. The classifier models must of course be trained, and some of these are also context-dependent. The work described in this paper may be regarded as part of a broader issue of how best to share task knowledge amongst the modules of a natural spoken dialog system.

6. REFERENCES