2009 Ninth IEEE International Conference on Data Mining

Automatically Extracting Dialog Models from Conversation Transcripts

Sumit Negi*, Sachindra Joshi*, Anup Chalamalla†, L Venkata Subramaniam*
sumitneg@in.ibm.com, jsachind@in.ibm.com, akchalam@uwaterloo.ca, lvsubram@in.ibm.com
*IBM Research - India
†University of Waterloo, Canada

***Abstract*—There is a growing need for task-oriented natural language dialog systems that can interact with a user to accomplish a given objective. Recent work on building task-oriented dialog systems have emphasized the need for acquiring task-specific knowledge from un-annotated conversational data. In our work we acquire task-specific knowledge by defining *sub-task* as the key unit of a task-oriented conversation. We propose an unsupervised, apriori like algorithm that extracts the sub-tasks and their valid orderings from un-annotated human-human conversations. Modeling dialogues as a combination of sub-tasks and their valid orderings easily captures the variability in conversations. It also provides us the ability to map our dialogue model to AIML constructs and therefore use off-the-shelf AIML interpreters to build task-oriented chatbots. We conduct experiments on real world data sets to establish the effectiveness of the sub-task extraction process. We codify the extracted sub-tasks in an AIML knowledge base and build a chatbot using this knowledge base. We also show the usefulness of the chatbot in automatically handling customer requests by performing a user evaluation study.**

I. Introduction

Natural language dialog systems are gaining in importance. In general, dialog systems are of two types: (1) open domain dialog systems such as ELIZA [1] and ALICE [2] that aim to engage in open ended conversations and (2) task oriented dialog systems [1] [2] involving completion of objectives such as pizza ordering, car booking and hotel reservation. In an open domain dialog system, the objective of the system is to present human-like responses to a user. On the other hand a task oriented dialog system not only needs to present human-like responses but also needs to ensure that the dialog objective is achieved by completing various sub-tasks. As an example, a dialog system for the car rental domain needs to perform various sub-tasks such as collect details about the time and date for pick up and drop off, and gather car preferences for completing the objective of car booking.

For task oriented dialog systems, the task/domain specific dialog information such as steps in a task and domain keywords need to be provided. Specifying this information manually is a time consuming process. Moreover, the hand-crafted knowledge may be brittle and rigid [3] and may not reflect the users perception of a task [4]. Therefore, there is merit in extracting this information from a collection of in-domain human-human conversations. In this paper, we propose a method to extract domain specific task structure from a real world un-annotated corpus of human-human conversation. Our task-structure model can be easily mapped to AIML[3] constructs. This provides us the ability to encode the extracted task structure in an AIML knowledge base with very little supervision. Existing AIML intepreters can then be used to create task oriented chatbots.

Unlike the work proposed in [5], we do not assume that the task structure is pre-specified and focus on learning this from the data itself. A task may contain various sub-tasks. As an example Figure 1 describes five sub-tasks that need to be completed for the task of 'Reserve a car'. Note that all the sub-tasks need to be executed to complete the task. Learning of the task structure is compounded by the fact that there could be various valid orderings of sub-tasks that can be adopted to perform the same task. Figure 1 shows various valid orderings in which these sub-tasks can be executed for performing the task of 'Reserve a Car'. In our work, we extract all the valid orderings of sub-tasks from the data instead of fixing a particular ordering as this truly models the actual flow of conversations as against a pre-defined ordering of sub-tasks. This is in contrast to the work proposed in [6] where only a fixed ordering of sub-tasks is considered. We introduce the notion of *pre-conditions* that specifies the necessary conditions for initiating a sub-task. These pre-conditions impose a set of valid orderings over the sub-tasks.

Task: Reserve a Car	Valid Sub-Task Ordering
Sub-Task 1: Ascertain Pickup and drop off location	• 1 → 2 → 3 → 4 → 5
Sub-Task 2: Ascertain Pickup and drop off time	• 1 → 3 → 2 → 4 → 5
Sub-Task 3: Gather car preference	• 3 → 1 → 2 → 4 → 5
Sub-Task 4: Gather Payment Details	• 3 → 2 → 1 → 4 → 5
Sub-Task 5: Confirm Booking	

Figure 1. Sub-Tasks and their Valid Orderings

Figure 2 provides an overview of the proposed method. In order to extract sub-tasks from a corpus, we first normalize individual utterances. This is quite important as natural

The work was done while Anup was with IBM Research - India
[1]http://www-ai.ijs.si/
[2]http://alice.pandorabots.com/
[3]http://alicebot.blogspot.com/

1550-4786/09 $26.00 © 2009 IEEE
DOI 10.1109/ICDM.2009.113

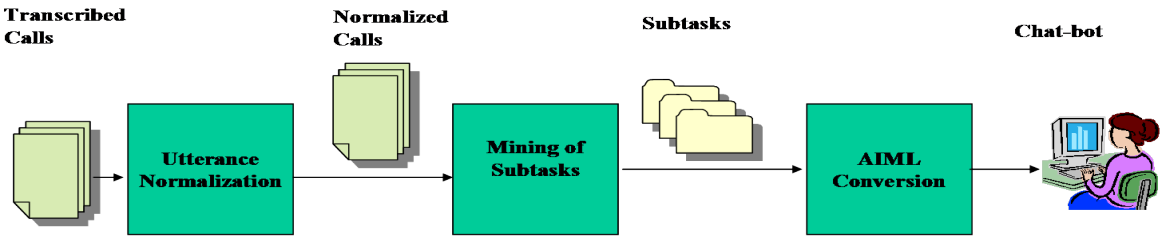

Figure 2. Overview of the Proposed Method

languages provide different ways to express the same thing. As an example, *can i have your credit card please* and *what is your card details* both can be used to gather payment details. Utterance normalization involves discovering a set of special features that are used to cluster semantically similar utterances. These special features are discovered by finding frequently occurring word patterns that may include gaps. Each call is then represented as a sequence of utterance clusters and frequent pattern mining is performed to discover sub-tasks and their valid orderings. The extracted task structure is encoded as an AIML knowledge base. An off-the-shelf AIML interpreter is then used to build a task-oriented chat-bot.

The key contributions of this paper are:

1) We model the domain specific task structure as a combination of sub-tasks and associated pre-conditions. The set of pre-conditions loosely define the valid orderings of sub-tasks.
2) We propose an efficient unsupervised method of finding sub-tasks and pre-conditions from human-human conversations.
3) We show how the extracted task structure can easily be encoded as an AIML construct to build a task-oriented chat-bot

The rest of the paper is organized as follows. We describe the utterance normalization in Section III. In Section IV we describe the method adopted for mining the task-structure from call transcripts. In Section V we provide details of how the extracted task-structure can be encoded into AIML constructs. In Section VI, we provide experimental results and details of the user study. Finally, we present our conclusion in Section VII.

II. Related Work

Acquiring task-specific knowledge using data-driven techniques for configuring dialog systems is a relatively new research area. Previously, supervised learning approaches have been used to acquire a task model [7], [2]. The work done by [3] for automatically creating dialog models using dialog act and task/sub-task information is also worth mentioning. Their approach involves the use of a dialog corpus which has been manually labeled with hierarchical information.

Several methods have been proposed for building spoken dialogue systems that allow a human user to interact with a machine using voice as the primary communication medium [8]. However, the problem of building spoken dialogue systems have additional problems of speech understanding and speech generation that fall outside the scope of our work.

Other data-driven approaches include the CU's dialog manager [9] and the AMITIES project [10]. The AMITIES project aims at building a dialog system from conversation transcripts. The "Frame Agents", implemented in the AMITIES dialog manager, handles tasks such as verifying customer's identity, *etc*. This is similar to our idea of *sub-tasks*. However, in their case the Frame Agent definitions are pre-specified.

Chotimongkol and Rudincky [6] proposed a method for acquiring domain specific information from task oriented and unannotated human-human conversations. They use structural representation (form based dialog structure) and use existing machine learning methods to acquire various components of this representation such as *concepts*, *sub-tasks* and *tasks*. However, they do not focus on extracting the valid ordering of sub-tasks that can be adopted for achieving a given task. In our work we not only extract, in an unsupervised manner, the sub-tasks, but also their valid orderings from un-annotated data. To the best of our knowledge our work of automatically building a task-oriented chat-bot from unannotated in-domain dialogs is a first.

III. Utterance Normalization

As pointed out in the Introduction section, natural languages provide different ways to express the same thing. As an example, Figure 3 shows various ways in which a customer can express his intention of picking up a car from a particular location. We need to cluster and canonicalize the semantically similar utterances so that they are referred to by their canonical form or simply a cluster label. We first replace all the named entities by their types and then generate a feature vector for each utterance. The feature vectors are then clustered to group together semantically similar utterances. In the following subsections we first describe the named entity annotator that we use and then discuss clustering of utterances.

I would like to pick the car at **SFO Airport**.
Can I pick it up at the **City Center**.
I want to pick up the car from **Wisconsin Avenue**.

Figure 3. Examples of Similar Utterances

A. Name Entity Annotation

We employ a rule-based named entity annotator that uses a set of dictionaries and rules to annotate various types of entities in the transcript data. The types of entities that we annotate are LOCATION, DATE, TIME, AMOUNT. We also handle a few domain specific named-entities such as CAR-MAKE (Chevrolet,Toyota), and VEHICLE-TYPE (car, van). All entities that are annotated in an utterance are replaced by their corresponding type.

B. Utterance Clustering

For each utterance, we generate a feature vector. We use n-grams ($n \leq 2$) that appear in an utterance as features. However, using only n-gram features for clustering utterances does not work well. As an example consider the following three utterances: (1) *do you have a valid credit card*, (2) *do you have a valid AAA member card* and (3) *do you have your credit card*. While these utterances have a good number of n-grams in common, they are not semantically similar. Using just n-grams would have grouped them together. Considering non-consecutive N-gram features such as valid#member#card and valid#credit#card we are able to distinguish these sentences as dissimilar. Where # signifies a gap of a few tokens.

Thus in addition to unigram and bigram features we use frequent non-consecutive word-sequences as features. These features are extracted from all utterances present in the transcripts by using the algorithm given in [11]. Agent and customer utterances are clustered separately. Weka's [4] K means clustering is used on the feature vectors to group together semantically similar utterances. For completeness, we present the method employed for generating frequent non-consecutive word sequences briefly in the following section. We also use the same method for discovering subtasks from normalized calls later in the paper.

1) Extraction of Frequent Non-Consecutive Item Sequences: In this section we describe a variation of apriori algorithm for association rule mining. Patterns that capture non-consecutive items (items could be words, entities, canonical question-cluster labels, *etc.*) Natural language expressions allow inclusion of complex modifiers. The intervening modifiers improve the richness of expressions and are thus important to natural languages. However, these variations make certain highly correlated words non-consecutive. Figure 4, presents a set of example sentences. All questions in Figure 4 are semantically similar, however they differ in their surface forms due to the use of modifiers and other intervening words.

[4]http://www.cs.waikato.ac.nz/ml/weka/

have you **rented** a **car** **before** from us.
have you **rented** a **car** from us **before**.
have you **rented** a **car** from <a_rental_agency>
before.

Figure 4. Example Similar Utterances

We implemented an efficient algorithm described in [11] (an extension of the apriori algorithm [12]) for finding frequent patterns of non-consecutive tokens. The $minSup$ value in apriori corresponds to the minimum number of times a non-consecutive n-gram should occur across all the sentences. So, the most frequent non-consecutive n-grams which exceed the threshold value of $minSup$ are output by the system. As an example, the following three non-consecutive patterns can be extracted from the utterances given in Figure 4:(1)rented#car#before (2)rented#car and (3)rented#before. The system considers only the longest sequence which is rented#car#before and leaves the rest.

Figure 5 describes the algorithm that discovers all the frequent sequences of length 1 for a user defined threshold N. The function $Sup(s_n)$ returns the support for the sequence s_n in the data. This is the fraction of documents in which the sequence s_n occurs, where a maximum token gap of (δ) is allowed between each subsequent token of the sequence s_n. The algorithm iteratively builds token-sequences of length $n+1$ from token sequences of length n. We refer the readers to [11] for the proof of optimality and further details.

Find $\mathcal{S}_1$, the set of all 1-item-sequences; $n = 1$
$\mathcal{S}_* = \{\}$
while $n \leq N$ **do**
  $\mathcal{S}_{n+1} = \{\}$
  for Each $s_n, s'_n \in \mathcal{S}_n$ **do**
    if s_n and s'_n have a subsequence of length $n-1$ in common **then**
      Merge s_n and s'_n to obtain s_{n+1}
      if $sup(s_{n+1}) \geq minSup$ **then**
        $\mathcal{S}_{n+1} = \mathcal{S}_{n+1} \cup \{s_{n+1}\}$
      end if
    end if
  end for
  for Each $s_n \in \mathcal{S}_n$ **do**
    if $\neg(\exists s_{n+1} \in \mathcal{S}_{n+1} | s_{n+1} \supset s_n)$ **then**
      $\mathcal{S}_* = \mathcal{S}_* \cup \{s_n\}$
    end if
  end for
  $n = n + 1$
end while

Figure 5. The algorithm for generating the set $\mathcal{S}_*$ of token-sequences with high support

IV. Mining of Sub-tasks from Call Transcripts

Using the utterance normalization process as described in Section III, calls are normalized i.e. each utterance in a

```
Utterance Pre-Conditions
CUST: #book#car.
CUST: #reserve#car#
CUST: #rent#car#
Flow Pre-Conditions
None
sub-task Template
AGENT: Which location would you like to pickup
the car?
AGENT: What date and time will you pick the car
up?
AGENT: What location would you drop the car off?
AGENT: What date and time would you return the
car?
```

Figure 6. Gather Pickup Information Sub-task

call is replaced by it's corresponding cluster label. In the following sections, we first describe how the sub-tasks are discovered from the normalized calls and then describe the method for finding out *pre-conditions* for each sub-task.

A. Finding Sub-tasks

As pointed out in the Introduction section, a sequence of sub-task needs to be executed to achieve the objective of a call. Since customers and agents often engage in similar kinds of interactions to accomplish an objective we mine frequent, possibly non-consecutive, utterance sequences. In our experiments, we observe that mining frequent non-consecutive agent and customer utterances does not give us very meaningful sub-tasks. This is because there is a lot of variability in customer utterances. On the contrary, agents follow a more or less standard sequence of utterances to accomplish a task. This could be attributed to the agents' training. Therefore, in order to discover sub-tasks, we look for only frequent non-consecutive agent utterances. We represent each call by the sequence of agent utterance cluster labels that occur in it and then use the algorithm given in Figure 5 for finding out frequent non-consecutive agent cluster sequences. These sequences are treated as sub-tasks. Note, that mining of sub-tasks can be thought of as mining 'vertical patterns' in transcripts. Figure 6 shows a sub-task extracted from the car-rental call transcripts.

B. Finding Pre-conditions for Sub-tasks

A sub-task defines a set of agent utterances that needs to be executed in order to perform a part of a task. We still need to find conditions, when a particular sub-task should be initiated. These conditions form the set of *pre-conditions* for a sub-task.

There are two types of pre-conditions for a sub-task. The first pre-condition, referred as *utterance pre-condition*, specifies utterances which when expressed by a customer indicates initiation of a particular sub-task. As an example, an utterance such as "please make this booking" from a customer is an indicator for the initiation of the "make payment" sub-task. Note, that customer utterances are important indicators for initiation of a sub-task as it is a customer who expresses particular intention that leads an agent to perform a particular action. The second pre-condition associated with a sub-task, referred as *flow pre-conditions* ensures that only logical flow of sub-tasks are allowed. As an example, the sub-task of "make payment" cannot be executed till the sub-task of "gather travel details" has been performed.

In order to discover *utterance pre-conditions* for a sub-task we collect all customer utterance clusters that appear just before a given sub-task from the normalized calls. We then take the frequent non-consecutive word sequence features, as generated in Section III, for each of these clusters. These word sequences form the *utterance pre-conditions* for the sub-task. Note, that there could be more than one *utterance pre-conditions* for a given sub-task. The *flow pre-conditions* are discovered by representing normalized calls as a sequence of sub-tasks. For a given sub-task, we collect all the sub-tasks that precede the sub-task. At least one of the sub-tasks from the collected set should have been executed in order to ensure the *flow pre-condition*. The flow pre-conditions can also be set manually if the number of sub-tasks is small. Figure 6 presents an example sub-task along with its *utterance pre-conditions* and *flow pre-conditions* that we obtain automatically for the car rental domain in our experiments.

V. AIML GENERATION

We now focus on encoding the extracted task structure in AIML [5]. AIML, or Artificial Intelligence Markup Language, is an XML dialect for creating natural language software agents.

A. Encoding Task Structure in AIML

For each agent utterance cluster in a given sub-task, a corresponding AIML *topic* tag is created. Every *topic* element in AIML has an associated *category* tag which in turn contains a *pattern* and a *template* tag. The *pattern* tag is used to ascertain *utterance pre-conditions*. The *template* tag is used to generate the corresponding agent utterances. In order to maintain the *flow pre-conditions* AIML's *think* tag is used. A *think* tag can be thought of as a state that provides a mechanism in AIML to manage conversational flow. Some sub-tasks may need to be integrated with back-end databases in order to complete the task. This is done using AIML's *system* tag. As an example, car availability needs to be checked in the back-end database once the sub-task "gather travel details" has been executed. Integration with the back-end database is performed using a program which embeds the required logic. We refer to this logic as *sub-task logic*. Note, that the *sub-task logic* integration is the only activity that needs to be done manually. Figure 7 provides an AIML representation for the "Gather Pickup Information" sub-task.

[5]http://www.alicebot.org/aiml.html

```
<topic>
<category>
<pattern>_ book_ car</pattern>
<template><srai>PICKUPLOC</srai></template>
</category>
<category>
<pattern>_ reserve_ car</pattern>
<template><srai>PICKUPLOC</srai></template>
</category>
<category>
<pattern>_ rent _ car *</pattern>
<template><srai>PICKUPLOC</srai></template>
</category>
<category>
<pattern>PICKUPLOC</pattern>
<template>
<think>
<set name="topic">TRAVELDETAIL1</set>
</think>
Which location would you like to pickup the car?
</template>
</category>
</topic>

</topic>
```

Figure 7. AIML code snippet for 'Gather Pickup Information' sub-task

VI. Evaluation Method and Experimental Results

This section describes the data-set and the experiments used to evaluate our method both at the component and system level. The first set of experiments measures the effectiveness of the task structure extraction process by providing results for the utterance normalization and the sub-task extraction steps. The second set of experiments involves a user-evaluation study that measures the usability of the AIML chat-bot built from the extracted task structure.

A. Data Set

In our experiments we used 975 manually transcribed calls from the car-rental domain. The average number of turns per call is 46, where a turn indicates a single dialog spoken either by an agent or a customer during the course of the call. The high number of turns in our data set is due to the fact that the conversation between agents and customers is often punctuated with protracted negotiations between customer and agents regarding car types, availability dates, offers etc. This make the job of extracting sub-tasks and their valid orderings challenging.

1) Utterance Normalization Evaluation: The Utterance Normalization step is evaluated by comparing the results of the Utterance Clustering against a manually tagged data set. A human expert was asked to manually categorize 266 utterances from the transcribed call data set into 5 utterance categories. These 5 categories are : "Pickup Location", "Pickup Date and Time","Membership or Discounts Details", "Price or Rate Negotiation","Vehicle Specification". In the experiments these clusters are referred to as A,B,C,D and E respectively. These 266 utterances are then normalized using the steps described in Section III. The normalized utterances were then clustered using CLUTO [6]. Table 1 reports the external cluster quality measures for these 5 clusters. As shown in Table 1, we achieve very high purity values for most of the clusters.

[6]http://glaros.dtc.umn.edu/gkhome/views/cluto/

Cluster	Utterances	Entropy	Purity
A	98	0.085	0.969
B	35	0.000	1.000
C	34	0.376	0.705
D	56	0.130	0.946
E	43	0.336	0.767
Total	Utterances	Avg. Entropy	Avg. Purity
5	266	0.161	0.877

Table I
Utterance Normalization Results

2) Sub-task Extraction Evaluation: The sub-task extraction step is evaluated by comparing the results of sub-task extraction with ground truth viz., manually marked sub-tasks in transcribed calls. Out of the 5 sub-tasks extracted by our sub-task extraction method a domain expert randomly selected two sub-tasks. He then manually assigned each utterance in 15 transcribed calls as belonging to one of the two sub-tasks. The two sub-tasks were "Gather pickup details" and "Gather membership details". As the sub-task extraction step resembles the call-segmentation task we borrow evaluation metrics from that domain. The quality of the sub-tasks extracted are measured using Segment Type Matching B_{JK} [13]. B_{JK} measures the fraction of sentences from the call that are mapped to the same *segment type* (in our case to the same sub-task) in both the automatic and manual segmentations. Our sub-task extraction experiment yielded a B_{JK} value of 0.80 (on a scale of 1), where values close to 1 are indicative of high sub-task extraction accuracy. Our B_{JK} value of 0.80 is close to what was achieved by [13] on a similar data-set.

3) From Dialog Model to chat-bot: The 5 sub-tasks extracted from the transcribed calls were converted into AIML knowledge base using the steps described in Section 5. The sub-task logic for each of the sub-tasks was manually integrated.

User input to the chat-bot is annotated using the named entity annotator before being passed to the AIML interpreter. The AIML interpreter tries to find the best match between the annotated user input and the *pattern* expressions of the various *categories* available in its AIML knowledge base. Once a *pattern* definition matches the user input the AIML interpreter executes the associated *template* definition. The template definition could be a simple response, a question or could involve the execution of some task logic (e.g. check the back-end database for car availability). Execution of the template definition by the AIML interpreter performs two major tasks. First, it extracts the values of task variables from the annotated user input. For instance, the LOCATION and DATE values are extracted if the input contains them and the corresponding task variables PICKUP_LOCATION and PICKUP_DATE are initialized.

These task variables may be required by other AIML

categories or may serve as inputs to task logic functions. For example task variables PICKUP_DATE, PICKUP_TIME, PICKUP_LOCATION, DROP_DATE serve as inputs to the *find-car-availability* sub-task logic. Second, a template may invoke the sub-task logic code using AIML's *system* tag. Based on the sub-task logic's return value the AIML interpreter may execute different dialog fragments (i.e. conditional branching). For our experiments we use Program D [7], a widely used AIML interpreter.

4) User Evaluation: An AIML compliant chat-bot was loaded with this knowledge base and 5 human evaluators were asked to evaluate the chat-bot system. None of the authors of this paper were amongst these evaluators. All evaluators were asked to converse with the chat-bot to accomplish the following 5 objectives: (1) Task A - Inquire price for booking a car from location X to Y, (2) Task B - Make a reservation from X to Y, (3) Task C - Make a reservation from location X to Y and claim AAA/Sam's club discount, (4) Task D - Modify the pickup location for an existing reservation, and (5) Task E - Modify the pickup date and time for an existing reservation.

The participants were required to fill out a questionnaire mentioning whether he/she was successful in completing the above five tasks. This input was collected after completion of individual tasks. User satisfaction was assessed by way of a questionnaire containing four statements. These covered ease of doing the task, how well the system understands the user inputs, how well the system works, and the users enjoyment of the system. Participants rated each on a five-point Likert scale. Summed results showed an average score of 17.75 over all users (higher score means greater satisfaction).

The results of the user study are shown in Figure 8. Tasks B and C have the lowest task completion rates i.e. 61% and 66% respectively. This is because these tasks involve a considerable amount of planning as compared to other tasks.

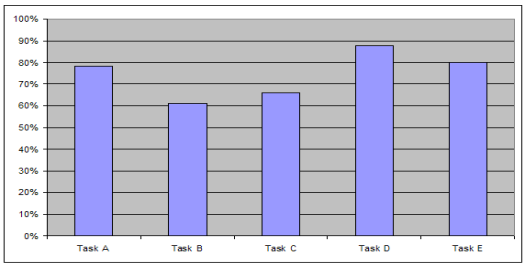

Figure 8. Results of User Evaluation Study

We investigated individual cases where the users were unable to complete the task and found that users frequently misspelt words or provided extra information than was asked by the chat-bot. These problems can be easily fixed by tweaking the automatically generated AIML.

[7]http://aitools.org/Program_D

VII. Conclusion

In this paper we presented a method to build a task-oriented conversational system from call transcript data in an unsupervised manner. We defined sub-tasks as the key unit of a task in a conversation and proposed a method to automatically discover them using an apriori like algorithm. We also introduced the notion of a set of *pre-conditions* for a sub-task that captures all the necessary conditions for initiating the sub-task. We mapped our dialog model to AIML constructs and used an off-the-shelf AIML interpreter to build a chat-bot. Our experimental evaluation demonstrated the efficacy of the proposed method.

References

[1] J. Allen, L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light, N. G. Martin, B. W. Miller, M. Poesio, and D. R. Traum., "The trains project: A case study in building a conversational planning agent," *Journal of Experimental and Theoretical AI*, 1995.

[2] J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift, and W. Taysom, "Plow: A collaborative task learning agent," in *AAAI*, 2007.

[3] S. Bangalore, G. D. Fabbrizio, and A. Stent, "Learning the structure of task-driven human-human dialogs," in *COLING/ACL*, 2006.

[4] N. Yankelovich., *Using Natural Dialogs as the Basis for Speech Interface Design*. Cambridge,MA: MIT Press, 1997.

[5] J. Alexandersson and N. Reithinger., "Learning dialogue structures from a corpus," in *EuroSpeech*, 1997.

[6] A. Chotimongkol and A. I. Rudnicky., "Acquiring domain-specific dialog information from task-oriented human-human interaction through an unsupervised learning," in *EMNLP*, 2008.

[7] J. Feng, S. Bangalore, and M. Rahim., "Webtalk:mining websites for automatically building dialog systems," in *ASRU*, 2003.

[8] B. Thomson, J. Schatzmann, and S. Young, "Bayesian update of dialogue state for robust dialogue systems," in *ICASSP*, 2008.

[9] W. Ward and B. Pellom., "The cu communicator system," in *ASRU*, 1999.

[10] H. Hardy, T. Strzalkowski, M. Wu, C. Ursu, N. Webb, A. Biermann, R. B. Inouye, and A. McKenzie., "Data-driven strategies for an automated dialogue system," in *CIKM*, 2005.

[11] A. Chalamalla, S. Negi, L. V. Subramaniam, and G. Ramakrishnan., "Identification of class specific discourse patterns." in *CIKM*, 2008.

[12] R. Agrawal and R. Srikant., "Fast algorithms for mining association rules." in *VLDB*, 1994.

[13] K. Kummamuru, Deepak P, S. Roy, and L. V. Subramaniam., "Unsupervised segmentation of conversational transcripts," in *SDM*, 2008.