Interactive Question Answering and Constraint Relaxation in Spoken Dialogue Systems

Sebastian Varges
CSLI
Stanford University
Stanford, CA 94305, USA
varges@stanford.edu

Fuliang Weng, Heather Pon-Barry
Research and Technology Center
Robert Bosch Corporation
4009 Miranda Ave, Palo Alto, CA, USA
`fuliang.weng, heather.pon-barry`
`@rtc.bosch.com`

Abstract

We explore the relationship between question answering and constraint relaxation in spoken dialog systems. We develop dialogue strategies for selecting and presenting information succinctly. In particular, we describe methods for dealing with the results of database queries in information-seeking dialogs. Our goal is to structure the dialogue in such a way that the user is neither overwhelmed with information nor left uncertain as to how to refine the query further. We present evaluation results obtained from a user study involving 20 subjects in a restaurant selection task.

1 Introduction

Information presentation is an important issue when designing a dialogue system. This is especially true when the dialogue system is used in a high-stress environment, such as driving a vehicle, where the user is already occupied with the driving task. In this paper, we explore efficient dialogue strategies to address these issues, and present implemented knowledge management, dialogue and generation components that allow cognitively overloaded users – see (Weng et al., 2004), for example – to obtain information from the dialogue system in a natural way. We describe a knowledge manager that provides factual and ontological information, a content optimizer that regulates the amount of information, and a generator that realizes the selected content. The domain data is divided between domain-specific ontologies and a database back-end. We use the system for both restaurant selection and MP3 player tasks, and conducted experiments with 20 subjects.

There has been substantial previous work on information presentation in spoken dialogue systems. (Qu and Green, 2002) also present a constraint-based approach to cooperative information dialogue. Their experiments focus on overconstrained queries, whereas we also deal with underconstrained ones. Moreover, we guide the user through the dialogue by making suggestions about query refinements, which serve a similar rôle to the conditional responses of (Kruijff-Korbayova et al., 2002). (Hardy et al., 2004) describe a dialogue system that uses an error-correcting database manager for matching caller-provided information to database entries. This allows the system to select the most likely database entry, but, in contrast to our approach, does not modify constraints at a more abstract level. In contrast to all the approaches mentioned above, our language generator uses overgeneration and ranking techniques (Langkilde, 2000; Varges and Mellish, 2001). This facilitates variation and alignment with the user utterance.

A long-standing strand of research in NLP is in natural language access to databases (Androutsopoulos et al., 1995). It mainly focused on mapping natural language input to database queries. Our work can be seen as an extension of this work by embedding it into a dialogue system and allowing the user to refine and relax queries, and to engage in clarification dialogs. More recently, work on question answering (QA) is moving toward *interactive* question answering that gives the user a greater role in the QA process (HLT, forthcoming). QA systems mostly operate on free text whereas we use a relational database. (Thus, one needs to 'normalize' the information contained in free text to use our implemented system without further adaption.)

In the following section, we give an overview of the dialogue system. We then describe the knowledge management, dialogue and generation components in separate sections. In section 6 we present evaluation results obtained from a user study. This is followed by a discussion section and conclusions.

2 System architecture

Our dialogue system employs the following architecture: the output of a speech recognizer (Nuance, using a statistical language model) is analyzed by both a general-purpose statistical dependency parser and a (domain-specific) topic classifier. Parse trees and topic labels are matched by the 'dialogue move scripts' of the dialogue manager (Mirkovic and Cavedon, 2005; Weng et al., 2005). The scripts serve to license the instantiation of dialogue moves and their integration into the 'dialogue move tree.' The use of dialogue move scripts is motivated by the need to quickly tailor the system to new domains: only the scripts need to be adapted, not the underlying machinery implemented in Java. The scripts define short sequences of dialog moves, for example a command move ("play song X") may be followed either by a disambiguation question or a confirmation that the command will be executed. A dialogue proceeds by integrating such scripted sequences into the dialogue move tree, yielding a relatively 'flat' dialogue structure.

Query constraints are built by dialogue move scripts if the parse tree matches input patterns specified in the scripts. These query constraints are the starting point for the processing strategies described in this paper. The dialogue system is fully implemented and has been used in restaurant selection and MP3 player tasks. There are 41 task-independent, generic dialogue move scripts, 52 restaurant selection scripts and 89 MP3 player scripts. The examples in this paper are mostly taken from the restaurant selection task.

3 Knowledge and Content management

The Knowledge Manager (KM) controls access to domain knowledge that is structured according to domain-dependent ontologies. The KM makes use of OWL, a W3C standard, to represent the ontological relationships between domain entities. The knowledge base can be dynamically updated with new instances at any point. In a typical interac-

tion, the Dialog Manager converts a user's query into a semantic frame (i.e., a set of semantic constraints) and sends this to the KM via the content optimizer. For example, in the Restaurant domain, a request such as "I want to find an inexpensive Japanese restaurant that takes reservations" results in the semantic frame below, where `Category` is a system property, and the other constraints are inherited properties of the Restaurant class:

```
(1) system:Category = restaurant:Restaurant
    restaurant:PriceLevel = 0-10
    restaurant:Cuisine = restaurant:japanese
    restaurant:Reservations = yes
```

In addition to the KM module, we employ a Content Optimization (CO) module that acts as an intermediary between dialogue and knowledge management during the query process. It receives semantic frames from the Dialogue Manager, revises the semantic frames if necessary (see below), and queries the Knowledge Manager.

The content optimizer also resolves remaining ambiguities in the interpretation of constraints. For example, if the user requests an unknown cuisine type, the otherwise often accurate classifier will not be able to provide a label since it operates under a closed-world assumption. In contrast, the general purpose parser may be able to provide an accurate syntactic analysis. However, the parse still needs to be interpreted by the content optimizer which has the domain-specific knowledge to determine that "Montenegrin restaurant" is a cuisine constraint rather than a service level constraint, for example. (See also section 7).

Depending on the items in the query result set, configurable properties, and (potentially) a user model, the CO module selects and performs an appropriate optimization strategy. To increase portability, the module contains a library of domain-independent strategies and makes use of external configuration files to tailor it to specific domains.

The CO module can modify constraints depending on the number of items in the result set, the system ontology, and information from a user model. Constraints can be relaxed, tightened, added or removed. The manner in which a constraint is modified depends on what kind of values it takes. For example, for the `Cuisine` constraint, values are related hierarchically (e.g., Chinese, Vietnamese, and Japanese are all subtypes of Asian), whereas `PriceLevel` values are linear (e.g., cheap, moderate, expensive), and `acceptsCreditCards` values are binary (e.g., ac-

cepted or not accepted).

If the original query returns no results, the content optimizer selects a constraint to modify and then attempts to relax the constraint value. If relaxation is impossible, it removes the constraint instead. Constraint relaxation makes use of the ontological relationships in the knowledge base. For example, relaxing a `Cuisine` constraint entails replacing it with its parent-concept in the domain ontology. Relaxing a linear constraint entails replacing the current value with an adjacent value. Relaxing a binary constraint entails replacing the current value with its opposite value.

Based on the ontological structures, the content optimizer also calculates statistics for every set of items returned by the knowledge manager in response to a user's query. If the result set is large, these figures can be used by the dialogue manager to give meaningful responses (e.g., in the MP3 domain, "There are 85 songs. Do you want to list them by a genre such as Rock, Pop, or Soul?").

The content optimizer also produces constraints that represent meta-knowledge about the ontology, for example, in response to a user input "What cuisines are there?":

```
(2) rdfs:subClassOf = restaurant:Cuisine
```

The processing modules described in the next sections can use meta-level constraints in similar ways to object-level constraints (see (1)).

4 Dialogue strategies for dealing with query results

In the following two sections, we describe how our dialogue and generation strategies tie in with the choices made by the content optimizer. Consider the following discourse-initial interaction for which the semantic frame (1) is constructed:

```
(3)
U: i want to find an inexpensive Japanese
   restaurant that takes reservations
S: I found 9 inexpensive Japanese
   restaurants that take reservations .
   Here are the first few :
S: GINZA JAPANESE RESTAURANT
S: OKI SUSHI CAFE
S: YONA SUSHI
S: Should I continue?
```

The example query has a relatively small result set which can be listed directly. This is not always the case, and thus we need dialogue strategies that deal with different result set sizes. For example, it does not seem sensible to produce "I found 2500 restaurants. Here are the first few: ...". At what

point does it become unhelpful to list items? We do not have a final answer to this question – however, it is instructive that the (human) wizard in our data collection experiments did not start listing when the result set was larger than about 10 items. In the implemented system, we define dialogue strategies that are activated at adjustable thresholds.

Even if the result set is large and the system does not list any result items, the user may still want to see some example items returned for the query. This observation is based on comments by subjects in experimental dry-runs that in some cases it was difficult to obtain any query result at all. For example, speech recognition errors may make it difficult to build up a sufficiently complex query. In response to this, we always give some example items even if the result set is large. (An alternative would be to start listing items after a certain number of dialogue turns.) Furthermore, the system should encourage the user to refine the query by suggesting constraints that have not been used yet. This is done by maintaining a list of constraints in the generator that is used up as the dialogue progresses. This list is roughly ordered by how likely the constraint will be useful. For example, using cuisine type is suggested before proposing to ask for information about reservations or credit cards.

In our architecture, information flows from the CO module to the generator (see section 5) via the dialogue move scripts of the dialogue manager. These are conditioned on the size of the *final* result set and whether or not any modifications were performed. Table 1 summarizes the main dialogue strategies. These dialogue strategies represent implicit confirmations and are used if NLU has a high confidence in its analysis of the user utterance (see (Varges and Purver, 2006) for more details on our handling of robustness issues). Small result sets up to a threshold t_1 are listed in a single sentence. For medium-sized result sets up to a threshold t_2, the system starts listing immediately. For large result sets, the generator shows example items and makes suggestions as to what constraint the user may use next. If the CO module performs any constraint modification, the first, constraint realizing sentence of the system turns reflects the modification. ('NP-original' and 'NP-optimized' in table 1 are used for brevity and are explained in the next section.)

30

	$|result_{final}|$	mod	example realization	f_{exp}
s1a	0	no	I'm sorry but I found no restaurants on Mayfield Road that serve Mediterranean food.	0
s1b	0	yes	I'm sorry but I found no [NP-original]. I did not even find any [NP-optimized].	0
s2a	small: $> 0, < t_1$	no	There are 2 cheap Thai restaurants in Lincoln in my database: Thai Mee Choke and Noodle House.	61
s2b	small	yes	I found no cheap Greek restaurants that have a formal dress code but there are 4 inexpensive restaurants that serve other Mediterranean food and have a formal dress code in my database:	0
s3a	medium: $>= t_1, < t_2$	no	I found 9 restaurants with a two star rating and a formal dress code that are open for dinner and serve French food. Here are the first ones:	212
s3b	medium	yes	I found no [NP-original]. However, there are N [NP-optimized]. Here are the first few:	5
s4a	large: $>= t_2$	no	I found 258 restaurants on Page Mill Road, for example Maya Restaurant , Green Frog and Pho Hoa Restaurant. Would you like to try searching by cuisine?	300
s4b	large	yes	I found no [NP-original]. However, there are N [NP-optimized]. Would you like to try searching by [Constraint]?	16

Table 1: Dialogue strategies for dealing with query results (last column explained in sec. 6)

5 Generation

The generator produces turns that verbalize the constraints used in the database query. This is important since the system may miss or misinterpret constraints, leading to uncertainty for the user about what constraints were used. For this reason, a generic system response such as "I found 9 items." is not sufficient.

The input to the generator consists of the name of the dialogue move and the relevant instantiated nodes of the dialogue move tree. From the instantiated move nodes, the generator obtains the database query result including information about query modifications. The core of the generator is a set of productions[1] written in the Java Expert System Shell (Friedman-Hill, 2003). We follow the bottom-up generation approach for production systems described in (Varges, 2005) and perform mild overgeneration of candidate moves, followed by ranking. The highest-ranked candidate is selected for output.

Productions map individual database constraints to phrases such as "open for lunch", "within 3 miles" and "a formal dress code", and recursively combine them into NPs. This includes the use of coordination to produce "restaurants with a 5-star rating and a formal dress code", for example. The NPs are integrated into sentence templates, several of which can be combined to form an output candidate turn. For example, a constraint realizing template "I found no [NP-

---

[1]Productions are 'if-then' rules that operate over a shared knowledge base of facts.

---

original] but there are [NUM] [NP-optimized] in my database" (see below for further explanation) can be combined with a follow-up sentence template such as "You could try to look for [NP-constraint-suggestion]".

The selection of which sentence template to use is determined by the dialogue move scripts. Typically, a move-realizing production produces several alternative sentences. On the other hand, the NP generation rules realize constraints regardless of the specific dialogue move at hand. This allows us to also use them for clarification questions based on constraints constructed from classifier information if the parser and associated parse-matching patterns fail; all that is required is a new sentence template, for example "Are you looking for [NP]?". We currently use 102 productions overall in the restaurant and MP3 domains, 38 of them to generate NPs that realize 19 possible input constraints (for both domains).

The decision of the CO module to relax or remove constraints also affects the generator: there are two sets of constraints, an 'original' one directly constructed from the user utterance, and an 'optimized' one used by the KM module to obtain the query result (see section 3). In case of constraint modifications, these two sets are not identical but often overlapping. To avoid generating separate sets of NPs independently for the two constraint sets, we assign unique indices to the constraints and hand the generator two index sets as targets of NP generation. We overgenerate NPs and check their index sets before integrating them into sentence templates.

turn	speaker	utterance
1.1	User	i need a restaurant with a five star food rating excellent service
1.2	System	I found 425 restaurants with a five star rating and excellent service , for example EL CAPITAN FISH MARKET , K-NINE COUNTRY CLUB , and KAISERHOF GERMAN AMERICAN . You could try to look for cuisine type .
1.3	User	which restaurants take reservations
1.4	System	I found 353 restaurants that have a five star rating , have excellent service and take reservations , for example GRAND CENTRAL CAFE , CHASIN CHICKEN , and CHON WOO RESTAURANT . Would you like to try searching by price ?

Figure 1: Example dialogue 1 from system evaluation using restaurant selection task

Ranking of candidate output moves is done by using a combination of factors. First, the ranker computes an alignment score for each candidate, based on its ngram-based overlap with the user utterance. For example, this allows us to prefer "restaurants that serve Chinese food" over "Chinese restaurants" if the user used a wording more similar to the first. We note that the Gricean Maxim of Brevity, applied to NLG in (Dale and Reiter, 1995), suggests a preference for the second, shorter realization. However, if the user thought it necessary to use "serves", maybe to avoid confusion of constraints or even to correct an earlier mislabeling, then the system should make it clear that it understood the user correctly by using those same words, thus preferring the first realization. Mild overgeneration combined with alignment also allows us to map the constraint `PriceLevel=0-10` in example (1) above to both "cheap" and "inexpensive", and use alignment to 'play back' the original word choice to the user. As these examples show, using alignment for ranking in NLG allows one to employ overgeneration techniques even in situations where no corpus data is available.[2]

Second, ranking uses a variation score to 'cycle' over sentence-level paraphrases. In the extreme case of repeated identical user inputs, the system simply chooses one paraphrase after the other, and starts over when all paraphrases have been used.

Third, we use an ngram filter based on bad examples ngrams, removing, for example, "Chinese cheap restaurants" but keeping "cheap Chinese restaurant." For generalization, we replace constraint realizations with semantic tags derived from the constraint names (except for the head noun), for example the trigram 'CUISINE PRICE restaurants'. An alternative is to use a more complex grammar formalism to prevent ungrammatical candidate moves.

6 Evaluation

We conducted experimental studies involving 20 subjects in a MP3 player task and 20 subjects in a restaurant selection task. In the following, we concentrate on the restaurant selection task because it is more challenging for constraint handling and information presentation.

Each subject in the restaurant selection task was given 9 scenario descriptions involving 3 constraints. Subjects were instructed to use their own words to find a fitting restaurant. We use a back-end database of 2500 restaurants containing the following information for each restaurant: restaurant name, cuisine type, city and street names, service level, rating, whether they accept credit cards or reservations, price level, open hours, dress code, additional information (e.g. vegetarian food) and distance from current location. Some of these constraints have a fixed number of values (e.g. service level and dress code), whereas those taking named entities as values are obviously open-ended.

Figures 1 and 2 show two of the dialogues from the experiments. To exemplify the challenges the system is faced with, user turns are shown in the form of the output of the speech recognizer (we currently use the best ASR hypothesis as input to the parser). The task of the first dialogue (figure 1) was to find a restaurant that has the following constraints: [Reservations=yes, ServiceLevel:excellent, Rating=5star]. Despite some minor speech recognition errors, the system correctly builds a query for the two constraints provided by the user in (1.1.) and verbalizes these in (1.2) to show its understanding of the user input. It also gives some example items and makes a suggestion. This suggestion is not taken up by the user

---

[2]However, we do have wizard-of-oz data to inform the system design (see section 7).

turn	speaker	utterance
2.1	User	breakfast
2.2	System	Are you looking for a restaurant that is open for breakfast ?
2.3	User	yes located on bower street
2.4	System	I found 71 restaurants on bower street that serve breakfast , for example , SOUPLANTATION-TO-GO , RAMAYANI WESTWOOD , and , PHILLY WEST . You could try to look for cuisine type .
2.5	User	which
2.6	System	I'm sorry, I do not understand what you mean by: which
2.7	System	Please try rephrasing that.
2.8	User	casual dress
2.9	System	There are 51 restaurants on bower street that serve breakfast and have a casual dress code in my database , for example , BERTY'S , LABOTETEGA , and , JAVAN RESTAURANT . Would you like to try searching by price ?
2.10	User	british
2.11	System	I found no restaurants on bower street that serve british food , have a casual dress code and are open for breakfast but there are 8 restaurants on bower street that are open for breakfast , serve European food and have a casual dress code in my database . Here are the first few :
2.12	System	HOTEL BEL-AIR

Figure 2: Example dialogue 2 from system evaluation using restaurant selection task

– in fact, we find that suggestions are generally not taken up by the user. We believe this is due to the nature of the tasks, which specified exactly which criteria to match. On the other hand, in more open application scenarios, where users may not know what questions can be asked, suggestions may be useful. In (1.3) the user issues a sub-query that further constrains the result set. By again summarizing the constraints used, the system confirms in (1.4) that it has interpreted the new constraint as a revision of the previous query. The alternative is to start a new query, which would be wrong in this context.

The task of the second dialogue, figure 2, was to find a restaurant that meets the constraints [BusinessHours:breakfast, StreetName='bower street', DressCode=casual]. This user tends to give shorter, keyword-style input to the system (2.1, 2.8). In (2.3), the user reacts to a clarification question and adds another constraint which the system summarizes in (2.4). (2.5) is an ASR error which the system cannot handle (2.6, 2.7). The user constraint of (2.8) is correctly used to revise the query (2.9), but "british" (2.10) is another ASR error that leads to a cuisine constraint not intended in the scenario/by the user. This additional constraint yields an empty result set, from which the system recovers automatically by relaxing the hierarchically organized cuisine constraint to "European food". In (2.11) the system uses dialogue

strategy s3b for medium-sized result sets with constraint modifications (section 4). The result of both dialogues is that all task constraints are met.

We conducted 20 experiments in the restaurant domain, 2 of which were restarted in the middle. Overall, 180 tasks were performed involving 1144 user turns and 1818 system turns. Two factors contributing to the higher number of system turns are a) some system turns are counted as two turns, such as 2.6, 2.7 in figure 2, and b) restaurants in longer enumerations of result items are counted as individual turns. On average, user utterances are significantly shorter than system utterances (4.9 words, standard deviation $\sigma = 3.82$ vs 15,4 words, $\sigma = 13.53$). This is a result of the 'constraint summaries' produced by the generator. The high standard deviation of the system utterances can be explained by the above-mentioned listing of individual result items (e.g. utterance (2.12) in figure 2).

We collected usage frequencies for the dialogue strategies presented in section 4: there was no occurrence of empty final result sets (strategy s1a/b) because the system successfully relaxed constraints if it initially obtained no results. Strategy s2a (small result sets without modifications) was used for 61 inputs, i.e. constraint sets constructed from user utterances. Strategy s3a/b (medium-sized result sets) was used for 217 times and required constraint relaxations in 5 cases. Strategy s4a/b (large result sets) was used for

316 inputs and required constraint relaxations in 16 cases. Thus, the system performed constraint modifications in 21 cases overall. All of these yielded non-empty final result sets. For 573 inputs, no modification was required. There were no empty final result set despite modifications.

On average, the generator produced 16 output candidates for inputs of two constraints, 160 candidates for typical inputs of 3 constraints and 320 candidates for 4 constraints. Such numbers can easily be handled by simply enumerating candidates and selecting the 'best' one.

Task completion in the experiments was high: the subjects met all target constraints in 170 out of 180 tasks, i.e. completion rate was 94.44%. An error analysis revealed that the reasons for only partially meeting the task constraints were varied. For example, in one case a rating constraint ("five stars") was interpreted as a service constraint by the system, which led to an empty result set. The system recovered from this error by means of constraint relaxation but the user seems to have been left with the impression that there are no restaurants of the desired kind with a five star rating.

7 Discussion

Based on wizard-of-oz data, the system alternates specific and unspecific refinement suggestions ("You could search by cuisines type" vs "Can you refine your query?"). Furthermore, many of the phrases used by the generator are taken from wizard-of-oz data too. In other words, the system, including the generator, is informed by empirical data but does not use this data directly (Reiter and Dale, 2000). This is in contrast to generation systems such as the ones described in (Langkilde, 2000) and (Varges and Mellish, 2001).

Considering the fact that the domain ontology and database schema are known in advance, it is tempting to make a closed world assumption in the generator (which could also help system development and testing). However, this seems too restrictive: assume, for example, that the user has asked for Montenegrin food, which is an unknown cuisine type, and that the statistical parser combined with the parse-matching patterns in the dialogue manager has labeled this correctly. The content optimization module will remove this constraint since there is no Montenegrin restaurant in the database. If we now want to generate "I did not find any restaurants that serve Montenegrin food

...", we do need to be able to use generation input that uses unseen attribute-value pairs. The price one has to pay for this increased robustness and flexibility is, of course, potentially bad output if NLU mislabels input words. More precisely, we find that if any one of the interpretation modules makes an open-world assumption, the generator has to do as well, at least as long as we want to verbalize the output of that module.

7.1 Future work

Our next application domain will be in-car navigation dialogues. This will involve dialogues that define target destinations and additional route planning constraints. It will allow us to explore the effects of cognitive constraints due to changing driving situations on dialogue behavior. The navigation domain may also affect the point of interaction between dialogue system and external devices: we may query a database to disambiguate proper names such as street names as soon as these are mentioned by the user, but start route planning only when all planning constraints are collected.

An option for addressing the current lack of a user model is to extend the work in (Cheng et al., 2004). They select the level of detail to be communicated to the user by representing the driver's route knowledge to avoid repeating known information.

Another avenue of future research is to automatically learn constraint relaxation strategies from (appropriately annotated) evaluation data. User modeling could be used to influence the order in which refinement suggestions are given and determine the thresholds for the information presentation moves described in section 4.

One could handle much larger numbers of generation candidates either by using packing (Langkilde, 2000) or by interleaving rule-based generation with corpus-based pruning (Varges and Mellish, 2001) if complexity should become an issue when doing overgeneration.

8 Conclusions

We described strategies for selecting and presenting succinct information in spoken dialogue systems. Verbalizing the constraints used in a query is crucial for robustness and usability – in fact, it can be regarded as a special case of providing feedback to the user about what the system has heard and understood (see (Traum, 1994), for example).

The specific strategies we use include 'backing-off' to more general constraints (by the system) or suggesting query refinements (to be requested explicitly by the user). Our architecture is configurable and open: it can be parametrized by empirically derived values and extended by new constraint handling techniques and dialogue strategies. Constraint relaxation techniques have widely been used before, of course, for example in syntactic and semantic processing. The presented paper details how these techniques, when used at the content determination level, tie in with dialogue and generation strategies. Although we focussed on the restaurant selection task, our approach is generic and can be applied across domains, provided that the dialogue centers around accessing and selecting potentially large amounts of factual information.

References

Ion Androutsopoulos, G.D. Ritchie, and P. Thanisch. 1995. Natural Language Interfaces to Databases – An Introduction. *Natural Language Engineering*, 1(1):29–81.

Hua Cheng, Lawrence Cavedon, and Robert Dale. 2004. Generating Navigation Information Based on the Driver's Route Knowledge. In *Proceedings of the Coling 2004 Workshop on Robust and Adaptive Information Processing for Mobile Speech Interfaces*, pages 31–38, Geneva, Switzerland.

Robert Dale and Ehud Reiter. 1995. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science*, 19:233–263.

Ernest Friedman-Hill. 2003. *Jess in Action: Java Rule-Based Systems*. Manning Publications.

Hilda Hardy, Tomek Strzalkowski, Min Wu, Cristian Ursu, Nick Webb, Alan Biermann, R. Bryce Inouye, and Ashley McKenzie. 2004. Data-driven strategies for an automated dialogue system. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 71–78, Barcelona, Spain, July.

forthcoming. *Proceedings of the workshop on Interactive Question Answering at HLT-NAACL 2006*.

Ivana Kruijff-Korbayova, Elena Karagjosova, and Steffan Larsson. 2002. Enhancing collaboration with conditional responses in information-seeking dialogues. In *Proc. of 6th workshop on the semantics and pragmatics of dialogue (EDILOG-02)*.

Irene Langkilde. 2000. Forest-based Statistical Sentence Generation. In *Proc NAACL-00*, pages 170–177.

Danilo Mirkovic and Lawrence Cavedon. 2005. Practical Plug-and-Play Dialogue Management. In *Proceedings of the 6th Meeting of the Pacific Association for Computational Linguistics (PACLING)*.

Yan Qu and Nancy Green. 2002. A Constraint-based Approach for Cooperative Information-Seeking Dialogue. In *Proceedings of the International Workshop on Natural Language Generation (INLG-02)*.

Ehud Reiter and Robert Dale. 2000. *Building Applied Natural Language Generation Systems*. Cambridge University Press, Cambridge, UK.

David Traum. 1994. *A Computational Theory of Grounding in Natural Language Conversation*. Ph.D. thesis, Computer Science Dept., U. Rochester.

Sebastian Varges and Chris Mellish. 2001. Instance-based Natural Language Generation. In *Proc. NAACL-01*.

Sebastian Varges and Matthew Purver. 2006. Robust language analysis and generation for spoken dialogue systems (short paper). In *Proceedings of the ECAI 06 Workshop on the Development and Evaluation of Robust Spoken Dialogue Systems*.

Sebastian Varges. 2005. Chart generation using production systems (short paper). In *Proc. of 10th European Workshop On Natural Language Generation*.

Fuliang Weng, L. Cavedon, B. Raghunathan, D. Mirkovic, H. Cheng, H. Schmidt, H. Bratt, R. Mishra, S. Peters, L. Zhao, S. Upson, E. Shriberg, and C. Bergmann. 2004. Developing a conversational dialogue system for cognitively overloaded users. In *Proceedings of the International Congress on Intelligent Transportation Systems (ICSLP)*.

Fuliang Weng, Lawrence Cavedon, Badri Raghunathan, Danilo Mirkovic, Ben Bei, Heather Pon-Barry, Harry Bratt, Hua Cheng, Hauke Schmidt, Rohit Mishra, Brian Lathrop, Qi Zhang, Tobias Scheideck, Kui Xu, Tess Hand-Bender, Stanley Peters, Liz Shriberg, and Carsten Bergmann. 2005. A Flexible Conversational Dialog System for MP3 Player. In *demo session of HLT-EMNLP 2005*.