

Comprehensive Exam

Topic: (Spoken) Dialogue Systems

Mihai Rotaru
Computer Science Department
University of Pittsburgh

1. Introduction

Interaction through language is a basic and ubiquitous human activity. Sometimes it is done in written form (books, newspapers, instant messaging). Sometimes it is done through speech and there are advantages to it: hands and eyes-free, better for conveying emotions, etc. Thus it is not surprising that we would like to interact with our tools through language. Dialogue systems are applications that allow a human to interact with a computer using natural language. If the interaction is via speech, they are called Spoken Dialogue Systems (**SDS**). While many things discussed here will generalize to dialogue systems in general, the focus here will be on SDS.

The scope of this write-up is to discuss some of the challenges in developing robust and efficient SDS. I will start by describing attempts to understand dialogue structure and its effect on dialogue management models. Next, I will discuss interaction design practices. I will talk about communication errors and error recovery techniques. Given the amount of interacting information sources in dialogue systems, I will also discuss statistical methods that improve performance by combining them. Finally, I will conclude by touching on a number of open issues.

2. Dialogue structure

A theory of dialogue structure will have to understand the role of speaker's utterances: why did the speaker utter an utterance and what is the utterance's effect on the conversation and its participants. Austin observed that there is more to an utterance than its linguistic form. He views every utterance as an action that manifests at three levels: locutionary, illocutionary and perlocutionary. The illocutionary act is of high interest to the dialogue community and was further refined by Searle and renamed speech acts. Dialogue acts (**DA**) are extensions of speech acts with more conversational functions motivated by practical dialogue experience. They are used to summarize the utterance semantic content and communication role. Many DA annotation schemes have been devised but most of them are domain and/or application dependent. The DAMSL (Dialogue Act Markup in Several Layers) annotation scheme proposed by (Core & Allen, 1997) is an attempt to devise a domain independent annotation scheme. While their inter-annotator reliability is not very high, their scheme is a good starting point for devising annotation schemes for domains of interest. For example, the DA scheme from (Stolcke et al., 2000) is an extension of DAMSL for spontaneous dialogues. Interestingly, (Walker, Passonneau, & Boland, 2001) apply DA tagging to system turns to improve their predictive models for user satisfaction.

DA tagging is the first level in automated discourse analysis. While it only offers a very coarse approximation of what happens in the conversation, it can enrich the information available for other tasks.

For example, DA tags can be used as additional information sources for automatic meeting summarization, to infer dialogue games and for measuring interactional dominance. In general, they are useful for tasks where a system is an observer of a human-human interaction and/or no deep understanding of the content of the interaction is needed. Human-human conversations exhibit certain conversational games that, while they might not serve a clear task purpose, are very frequent in human-human interaction and contribute to a more efficient interaction. For example, (Cohen, Giangola, & Balogh, 2004) note that it is important for dialogue systems to ground a user's negative response to a system offer. (Purver, Ginzburg, & Healey, 2003) analyze the clarification adjacency pairs and find correlations between the type of the clarification request and the type of answer. This type of behavior can be automatically induced by analysis of DA tagged corpora. Moreover, while for task-oriented conversation it can be argued that the dialogue structures follows loosely the task structure, DA tagging is a good starting point for understanding the structure of spontaneous dialogues.

A number of dialogue (discourse) structure theories have been proposed in the literature. Probably the most influential theory is the work by (Grosz & Sidner, 1986). They view the dialogue as a set of discourse segments organized in a tree structure based on their discourse segment purpose. As suggested by their theory, a task oriented dialogue will follow the task structure but it will enrich it with various conversation games (confirmation, clarifications, recall of previous segments, etc). While their theory does not explain all dialogue phenomena (e.g. backchanneling), their insights laid the foundation for expressive dialogue managers (Bohus & Rudnicky, 2003; Horvitz & Paek, 1999; Rich, Sidner, & Lesh, 2001). Many of this dialogue managers borrow concepts from (Grosz & Sidner, 1986) theory: hierarchical task representation, a stack (agenda) to maintain entities in focus, etc.

The complexity of the dialogue a SDS can engage in is dependent on the flexibility of its core component: the dialogue manager (**DM**). Several DM models have been proposed. The simplest one is the finite state DM (McTear, 1998). This model has a lot of limitations: allows only for system initiative, error recovery is complicated placing a lot of burden on the dialogue designer. All these issues limit the applicability of this DM model to only very simple tasks. The hierarchical decomposition of a dialogue in dialogue segments observed by (Grosz & Sidner, 1986) motivates the DM proposed by (Bohus & Rudnicky, 2003). Their model is an extension of form-based models and allows for a clear separation between task and discourse behavior. The biggest advantage of their approach is that conversational games (confirmations, generic dialogue abilities) can be implemented separately from the task in this way allowing the DM designer to focus only on the task specification. An even more powerful model is the plan-based model (Larsson & Traum, 2000; Rich et al., 2001). This model can explain phenomena observed in an interaction via update rules and thus can increase the complexity of system behavior by adding appropriate rules. But this flexibility is also a weakness: the interaction between rules present in the system can be very complex and can easily overwhelm even an experimented DM designer. Probabilistic models (Horvitz & Paek, 1999) can be used to deal with highly uncertain inputs. A hybrid model that will combine the intuitiveness of the form-based approaches, the flexibility of the plan-based approaches and the ability to handle uncertainty of the probabilistic approaches will prove to be a valuable asset to the SDS community.

3. Dialogue strategy design

Designing dialogue strategies involves many non trivial decisions. Usually, the SDS designer makes these decisions based on his intuition and experience but sometimes he fails to account for many factors (user population, ASR robustness, system limitations, etc). The SDS community has tried to address this problem by proposing methods to automate the process. The general idea is to view parts or the entire dialogue strategy as an optimization problem. There are several approaches to solve this optimization problem.

Viewing a dialogue as a Markov Decision Process (**MDP**) is one of them. In the MDP framework, design choices become policies and finding the best design choice corresponds to finding the optimal policy in a MDP. Previous work varies in what type of strategies they were optimizing. (Levin, Pieraccini, & Eckert, 2000) show that the entire dialogue structure can be optimized given a carefully chosen optimization function. But for any but the simplest tasks their approach is unfeasible due to a huge MDP state space. Since most of the higher level decisions (e.g. task structure) are usually easy to make, researchers have focused on learning low level aspects of system behavior. (Scheffler & Young, 2002; Singh, Litman, Kearns, & Walker, 2002) address the problem of initiative and confirmations and show that their optimized policies are better than hand-crafted policies.

There are several problems with MDP approaches to dialogue strategy design. The main disadvantage of viewing dialogue as a MDP is the amount of exploration that needs to be done to arrive to the optimal policy. Usually reinforcement learning is used to reduce the amount of exploration needed and in addition this method handles well delayed feedbacks. Many studies employ simulated user models to cope with the large amount of exploration (Levin et al., 2000; Scheffler & Young, 2002). But these user models are usually simple and policies devised based on them should be confirmed with additional real user studies. On the other hand, if the policy space is reduced to a subset that produces meaningful dialogues for any policy, real users can be used to guide the optimization process (Singh et al., 2002). Also, the MDP state space representation is very important due to the limitation imposed by the Markovian assumption and the exponential growth in the number of state attributes. By building a simulation of user behavior independent of state space representation, (Scheffler & Young, 2002) show that various state space representation can be compared (though their results show that the optimal policy cost was similar for all representations they experimented with). The cost function is also a very important choice in the MDP approach and sometimes the designer might not know from the beginning the best choice for this function. The cost functions usually balance several factors using a weight sum. (Scheffler & Young, 2002) show that the learned policy is highly dependent on the weights of the cost function; thus approaches that can test several cost functions are desirable.

Most of the optimization functions try to balance the dialogue length with dialogue quality. But sometimes the cost of operation can be an important drive behind the optimization process. (Paek & Horvitz, 2004) work on optimizing automated call routing is such an example. Their strategy choice is driven by cost (monetary) optimization and takes into account processes that happen outside the SDS (queuing models that approximate how call centers handles incoming calls).

The MDP approach to strategy design has problems dealing with history due to the Markovian assumption. Whenever history needs to be accounted for, the state space size will increase exponentially. Moreover, the optimal strategies learned using MDPs are “one-size-fits-all” strategies and can not be adapted to user expertise, task difficulty etc. Models that gather evidence from the interaction and use it to guide the strategy try to address these problems. (Chu-Carroll, 2000) shows how the cumulative effect of information extracted from previous user utterances can be used based on Dempster-Shafer theory of evidence to adapt the initiative strategy to the interaction so far. (Horvitz & Paek, 1999) show how the entire dialogue structure can be adapted. They use Bayesian networks to reason on the evidences available so far and value-of-information to guide what other input channels the system should probe or what the system should ask the user next.

User satisfaction plays an important factor in system design especially for commercial application. But it is not clear what factors influence user satisfaction and to what extent. The PARADISE framework (Walker, Litman, Kamm, & Abella, 2000) can be used to understand what parts of system behavior are of importance to users. Knowing these factors can help identify which components of a SDS system need to be improved. The PARADISE framework also has predictive properties: it can be used to predict user satisfaction without eliciting it from the user. This information can be used as a method to detect problematic dialogue and hand the call to a human operator before irritating the user.

4. Errors in interaction

Faced with a noisy input channel a SDS needs to have robust error recovery strategies. Many error recovery strategies are being explored but it is not yet clear which strategies are the most effective in a given situation. One way to recover from errors is to change the input modality. If the interaction is confined to telephone interaction, then the fall-back communication channel is the keypad. While keypad input is very reliable, it has many disadvantages. The most important one is the fact that the user has to change the modality (from speaking to pressing keys) which can be surprising, frustrating and non-intuitive. (Filisko & Seneff, 2004)'s analysis of such a system showed that in about 50% of the cases in which the system fell-back to the keypad entry the user did not type anything or hung up. Nonetheless, some types of information are more acceptable for keypad entry due to the fact that users have previous experience with transmitting that information via keypad (e.g. calendar date).

If the interaction is not confined to telephone interaction, multimodal systems can be used to take advantage of other modalities. Multimodal systems typically augment the speech channel with a pen-based channel that can be used for pointing, selecting and writing. Previous work (Oviatt et al., 2000) mentions that systems that employ both modalities perform better and offer better error handling than systems that use only one modality. By having more modalities, mutual disambiguation can be used to improve the error detection due to the presence of more correlated information sources. They also offer more user control/initiative in the sense that the user can select the right modality to solve the communication error (e.g. the user can use an on-screen keyboard if speech fails).

Another approach is to offer more functionality in the speech channel. (Filisko & Seneff, 2004) investigate the utility of using speak-and-spell for the MIT Mercury air travel system. The speak-and-spell recovery technique prompts the user to utter again the problematic word (in their study the departure/arrival city) and, in addition, utter the word spelling. By allowing for speak-and-spell functionality in a SDS, the user does not have to change the modality and thus it might be less taxing in terms of user satisfaction (spelling problematic words is a common human-human error recovery technique). Moreover, the uttered version of the word and the spelled version of the word are two complementary information sources and can be combined to improve recognition performance. But there are also problems. The user might have problem spelling long words or spelling complicated words (ex: Massachusetts). For this reason one might expect a lot of speech repairs in a spelling turn. Also, there is huge variability in spelling style and recognition accuracy for letters is pretty low. (Filisko and Seneff, 2004)'s study shows that there is potential for this technique, suggesting that this technique can be employed as an additional fall-back technique and used before falling back to keypad entry.

Speech errors are not the only errors a SDS should be able to recover from. Task errors are another example. They can be caused either by the system or by the user making a mistake. (Horvitz & Paek, 1999) incorporate in their Bayesian evidence processing an extra state that reasons whether the system has made a mistake and is in an incorrect subtask. A backtracking mechanism is used if the evidence for that states indicates a system task mistake. The situation when a user makes a mistake is also legitimate if we think about a SDS that teaches a user how to perform a task or when the user is unfamiliar with the task domain. The mistake made by a user can be harmless (its effect does not affect the conversation flow) or it can be harmful (a recovery strategy needs to be applied). (Garland, Lesh, & Rich, 2003) describe how detecting and recovering from user mistakes can be seamlessly integrated in a plan-based dialogue system.

Another way to reduce the number of communication errors is to maintain the mutual understanding between the system and the user. For the speech channel, this can be done via implicit or explicit confirmations. But the more confirmation the system does, the lengthier the dialogues are which can lead to a decrease in user satisfaction (Walker et al., 2001). MDP approaches can be used to determine the most efficient triggers for confirmation (Singh et al., 2002). For multi-modal systems, other modalities

(e.g. graphics) can be used to maintain common ground and address limitation in the speech channel and human memory limitations.

5. Integration of interacting information sources through statistical models

Many of the tasks a SDS has to solve in order to function properly have strong interconnection. For example, detecting speech repairs interacts with detection of intonational boundaries, POS tagging and speech recognition (Heeman & Allen, 1999). Because of these interactions, researchers have tried to solve these tasks together: instead of resolving each task individually in a pipeline fashion, the tasks are solved simultaneously taking advantage of the existing connection between the tasks. Statistical modeling is used to model these interactions.

The work of (Stolcke et al., 2000) applies this technique to dialogue act labeling. They combine the DA prediction task with word n-grams, decision trees, and neural networks modeling the idiosyncratic lexical and prosodic manifestations of each dialogue act. (Heeman & Allen, 1999) address the task of detecting and correcting speech repairs simultaneously with POS tagging, intonational phrase detection and discourse marker detection. (Filisko & Seneff, 2004) extend their SDS system with the speak-and-spell functionality that combines the uttered version of the word and the spelled version of the word to improve recognition. In all studies, their results show that as more information sources are added there is an improvement in prediction accuracy as defined by the study's evaluation method. Their results show that by solving their tasks simultaneously, they obtain better results on each task than addressing them separately. Unfortunately, when (Heeman & Allen, 1999; Stolcke et al., 2000) integrated their tasks with the speech recognition task the improvements for the speech recognition were in general very small.

One of the biggest disadvantage of solving tasks simultaneously is that the size of the training set is much larger than when tasks are resolved separately (due to the fact that we are removing the independence assumption). Consequently, the large annotation effort involved limits the applications of this technique.

6. Other open issues

While portability is a big challenge for computer science in general, the amount of reuse when porting a SDS to a new domain is very limited. Researchers have tried various methods to increase portability. A prerequisite for portability is flexible dialogue architectures. Hub-based architecture is an example of such architecture and they are widely used in the SDS community (Allen et al., 2000; Pellom, Ward, & Pradhan, 2000). They offer message-passing communication and SDS components can be easily and transparently replaced with new components. This allows flexibility in testing various versions of the same component.

Trying to delimit in SDS components the task independent part from the task dependent part is another way to increase reuse. Creating and/or using the right framework for that component are vital in these cases. For example, a state-based dialogue manager will have very limited reuse while dialogue managers like the one proposed by (Bohus & Rudnicky, 2003) will allow for a high degree of reuse. The latter dialogue manager provides task independent functionality (grounding behavior, turn taking and timing, generic dialogue mechanisms) allowing the DM designer to focus on task dependent behaviors.

Semi-supervised or unsupervised methods of components reuse are also important: being able to reuse training corpora from other domains/fields will be a great asset. For example, (Galescu, Ringger, & Allen, 1998) describe methods to rapidly build language models for new domains based on data from similar domains. There is also a huge amount of research done on monologues and text in general. Methods for using corpora or tools from that domain will speed the development of SDS. (Heeman & Allen, 1999) work on detecting and correcting speech repairs is one step towards this. Ignoring speech repairs will

require new ASR language models (thus lots of corpora need to be collected) and will also pose problem for parsing and language understanding. For parsing, grammars need to be extended to cover speech repairs while for language understanding the reparandum and the editing terms of a speech repair have to be replaced with the alteration. Detecting and correcting speech repairs earlier alleviates all these problems and potentially allows one to use parsing and understanding models trained on monologue corpora.

Also, creating frameworks, models and techniques that generalize and apply to different domains and tasks is one way to increase reuse. The PARADISE evaluation framework (Walker et al., 2000) is one such example. (Walker et al., 2000; Walker et al., 2001) have shown that the framework can be applied in different domains (e-mail system, train schedule system, voice dialing, air travel information systems). Moreover, some observations made in one domain regarding what factors affect user satisfaction seem to generalize to other domains too (e.g. task completion, mean recognition score).

The SDS community will benefit if offline corpora analysis methodologies are devised and tools for automating the analysis process will be available. For example, the ASR performance prediction proposed in (Litman, Hirschberg, & Swerts, 2000) can be used to find misrecognized turns without the need of transcribing the turns in the corpora. Also, the PARADISE framework (Walker et al., 2000) can be used to automatically detect problematic dialogues.

Even having a SDS with perfect recognition and understanding will still not be enough. The nature of the conversation places constraints on how much time a system has to react to a user turn. SDS architectures that allow parallelization like those described in (Allen et al., 2000; Pellom et al., 2000) are good approaches to address the real-time issues. Another way to buy time is for SDS to employ behavior used by humans in these situations. Tricks like holding the turn or using speech repairs in system turns are worth investigating.

While SDSs are very complex systems, the SDS community still lacks standards, corpora sharing and tools. To draw a parallel, a key to the success to graphical interfaces has been the development of standards and middleware packages. For example, in any graphical application users expect a help menu item or that they can access (sometimes context-sensitive) help by pressing F1. In the same way, one can imagine generic dialogue commands that every SDS should implement (like help, cancel, restart, etc). This will also have a positive effect on users since it will educate them about what things they can do when they are in trouble. Corpora and components sharing and availability can be of great benefit to the SDS community. For example, it is well known that there is large variability in what users say to correct the system errors and handling these corrections is a real challenge. But in most cases, a SDS designer will start the process from scratch since corpora on correction are not readily available or exploited. The availability of various practical tools can also increase the quality of new SDS. For example, a tool that detects whether there is speech or just noise on the speech channel will be helpful for a more robust handling of the barge-ins.

References

- Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., & Stent, A. (2000). An Architecture for a Generic Dialogue Shell. *Natural Language Engineering*, 6(3-4), 213-228.
- Bohus, D., & Rudnicky, A. (2003). RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. Paper presented at the Eurospeech.
- Chu-Carroll, J. (2000). MIMIC: An adaptive mixed initiative spoken dialogue system for information queries. Paper presented at the Applied Natural Language Processing.
- Cohen, M. H., Giangola, J. P., & Balogh, J. (2004). *Voice User Interface Design*. Boston: Addison-Wesley.
- Core, M., & Allen, J. (1997). Coding Dialogs with the DAMSL Annotation Scheme. Paper presented at the AAAI Fall Symposium on Communicative Action in Humans and Machines.

- Filisko, E., & Seneff, S. (2004). Error Detection and Recovery in Spoken Dialogue Systems. Paper presented at the HLT-NAACL Workshop on Spoken Language Understanding for Conversational Systems and Higher Level Linguistic Information for Speech Processing.
- Galescu, L., Ringger, E., & Allen, J. (1998). Rapid Language Model Development for New Task Domains. Paper presented at the Conference on Language Resources and Evaluation (LREC).
- Garland, A., Lesh, N., & Rich, C. (2003). Responding to and recovering from mistakes during collaboration. Paper presented at the IJCAI Workshop on Mixed-Initiative Intelligent Systems.
- Grosz, B., & Sidner, C. L. (1986). Attentions, intentions and the structure of discourse. *Computational Linguistics*.
- Heeman, P., & Allen, J. (1999). Speech repairs, intonational phrases and discourse markers: modeling speakers' utterances in spoken dialog. *Computational Linguistics*, 25(4).
- Horvitz, E., & Paek, T. (1999). A Computational Architecture for Conversation. Paper presented at the User Modeling.
- Larsson, S., & Traum, D. (2000). Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*(Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering).
- Levin, E., Pieraccini, R., & Eckert, W. (2000). A Stochastic Model of Human Machine Interaction for Learning Dialog Strategies. *IEEE Transactions on Speech and Audio Processing*, 8:1.
- Litman, D., Hirschberg, J., & Swerts, M. (2000). Predicting Automatic Speech Recognition Performance Using Prosodic Cues. Paper presented at the NAACL.
- McTear, M. F. (1998). Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit. Paper presented at the ICSLP.
- Oviatt, S., Cohen, P., Wu, L., Vergo, J., Duncan, L., Suhm, B., et al. (2000). Designing the user interface for multimodal speech and pen-based gesture applications: State-of-the-art systems and future research directions. *Human Computer Interaction*, 15(4), 263-322.
- Paek, T., & Horvitz, E. (2004). Optimizing Automated Call Routing by Integrating Spoken Dialog Models with Queuing Models. Paper presented at the HLT-NAACL.
- Pellom, B., Ward, W., & Pradhan, S. (2000). The CU Communicator: An Architecture for Dialogue Systems. Paper presented at the ICSLP.
- Purver, M., Ginzburg, J., & Healey, P. (2003). On the Means for Clarification in Dialogue. *Current and New Directions in Discourse and Dialogue*, 22, 235-255.
- Rich, C., Sidner, C. L., & Lesh, N. (2001). COLLAGEN: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4), 15-25.
- Scheffler, K., & Young, S. (2002). Automatic Learning of Dialogue Strategy using Dialogue Simulation and Reinforcement Learning. Paper presented at the HLT.
- Singh, S., Litman, D., Kearns, M., & Walker, M. (2002). Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research*(16).
- Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., et al. (2000). Dialogue Act Modeling for Automatic Tagging and Recognition of Conversational Speech. *Computational Linguistics*, 26:3.
- Walker, M., Litman, D., Kamm, C., & Abella, A. (2000). Towards Developing General Models of Usability with PARADISE. *Natural Language Engineering*.
- Walker, M., Passonneau, R., & Boland, J. (2001). Quantitative and Qualitative Evaluation of Darpa Communicator Spoken Dialogue Systems. Paper presented at the ACL.