Flexible Turn-Taking for Spoken Dialog Systems

Antoine Raux

CMU-LTI-08-xxx

December 2008

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Maxine Eskenazi, Chair
Alan W Black
Reid Simmons
Diane J. Litman, U. of Pittsburgh

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the U.S. National Science Foundation under grant number IIS-0208835

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

Keywords: Turn-Taking, Dialog, Dialog Systems, Speech, Computers, Science

To my wife Miyako,

Abstract

Even as progress in speech technologies and task and dialog modeling has allowed the development of advanced spoken dialog systems, the low-level interaction behavior of those systems remains often rigid and inefficient.

The goal of this thesis, is to provide a framework and models to endow spoken dialog systems with robust and flexible turn-taking abilities. To this end, we designed a new dialog system architecture that combines a high-level Dialog Manager (DM) with a low-level Interaction Manager (IM). While the DM operates on user and system turns, the IM operates at the sub-turn level, acting as the interface between the real time information of sensors and actuators, and the symbolic information of the DM. In addition, the IM controls reactive behavior, such as interrupting a system prompt when the user barges in. We propose two approaches to control turn-taking in the IM.

First, we designed an optimization method to dynamically set the pause duration threshold used to detect the end of user turns. Using a wide range of dialog features, this algorithm allowed us to reduce average system latency by as much as 22% over a fixed-threshold baseline, while keeping the detection error rate constant.

Second, we proposed a general, flexible model to control the turn-taking behavior of conversational agents. This model, the Finite-State Turn-Taking Machine (FSTTM), builds on previous work on 6-state representations of the conversational floor and extends them in two ways. First, it incorporates the notion of turn-taking action (such as grabbing or releasing the floor) and of state-dependent action cost. Second, it models the uncertainty that comes from imperfect recognition of user's turn-taking intentions. Experimental results show that this approach performs significantly better than the threshold optimization method for end-of-turn detection, with latencies up to 40% shorter than a fixed-threshold baseline. We also applied the FSTTM model to the problem of interruption detection, which reduced detection latency by 11% over a strong heuristic baseline.

The architecture as well as all the models proposed in this thesis were evaluated on the CMU Let's Go bus information system, a publicly available telephone-based dialog system that provides bus schedule information to the Pittsburgh population.

Acknowledgments

I would like to thank my advisor, Maxine Eskenazi for giving me guidance when I needed it while leaving me the freedom to pursue my own interest during the past 6 years. In addition to mentoring my research, Maxine has always been concerned about other aspects of my life, in particular my family, and I thank her for that. Alan W Black has been a kind of informal co-advisor for me during my whole PhD, and I am deeply grateful for the numerous hours he has spent helping me shape up and strengthen my research. I would also like to thank Diane Litman and Reid Simmons for agreeing to be on my thesis committee and providing many helpful comments and suggestions.

During the course of my PhD, I made some very good friends in Pittsburgh. More often than not, they offered both academic and emotional support. In particular, I would like to thank Satanjeev "Bano" Banerjee, Dan Bohus, Thomas Harris, Brian Langner, Mihai Rotaru, and Jahanzeb Sherwani for the many discussions we had, whether it be on the pros and cons of reinforcement learning for dialog management, or on the latest Steelers game.

I would also like to thank all the members of the Sphinx Group at CMU, of the Dialogs on Dialogs student reading group, and of the Young Researchers Roundtables on Spoken Dialog Systems participants, for many many discussions that helped me both define my research and broaden my horizon. Many thanks to Hua Ai, Tina Bennett, Anan-lada "Moss" Chotimongkol, Heriberto Cuayahuitl, Matthias Denecke, Bob Frederking, Hartwig Holzapfel, Matthew Marge, Jack Mostow, Ravi Mosur, Verena Reiser, Alex Rudnicky, Jost Schatzmann, Rich Stern, Svetlana Stoyanchev, and Jason Williams.

I would like to thank my family. My parents, my sister Cecile and her family, and my brother Xavier, and my grandmother Jeannette, for always trusting me to go my own way and supporting me, even if that meant going thousands of miles away from home, and then thousands of miles in the other direction.

Finally, my deepest gratitude goes to my children Yuma and Manon, and my wife Miyako. Thank you for bearing with me when deadlines meant more stress and less time at home. Thank you Manon for brightening my days with your smiles and laughter and

singing. Thank you Yuma for reminding me what really matters, with all the wisdom of your 4 years, and for building me a ”super special pen” that writes theses faster when I needed it. Last but, oh so not least, thank you Miyako for your love, patience, and strength in carrying the whole family during the past 6 years. I could not have done it without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

After several decades of research and development effort in the realm of practical spoken dialog systems, the technologies have matured enough to allow wide spread use of such systems. Still, the approaches that have permitted the creation of working systems have left many issues unsolved and spoken conversation with artificial agents remains often unsatisfactory. Perhaps the most prominent issue is the quality of automatic speech recognition (ASR), which often results in misunderstandings that, in turn, lead to dialog breakdowns. To overcome these issues system designers either constrain the interaction in some way, as is the case in system-directed dialogs [Farfán et al., 2003], or endow systems with error handling capabilities to smoothly recover from misrecognitions [Edlund et al., 2004, Bohus and Rudnicky, 2005]. These two strategies provide a way to cope with imperfect ASR, but they both come with a cost: they make dialogs longer either by only letting the user provide small amounts of information at a time (as in strongly system-directed dialogs), or by generating confirmation prompts (as in systems with error handling strategies). This would not be an issue if, in addition to issues in spoken language understanding, current spoken dialog systems did not also have poor turn-taking capabilities. Indeed, the cost of an additional turn for artificial conversational agents, in time spent and/or disruption of the flow of the conversation, is much higher than what happens in human-human conversation. As pointed out in recent publications [Porzel and Baudis, 2004, Ward et al., 2005], this weakness comes from the fact that low-level interaction has to a large extent been neglected by researchers in the field. Instead, they have concentrated on higher-level concerns such as natural language understanding and dialog planning.

1.2 The Conversational Floor

Before going into any more details on turn-taking and system dialog systems, it is important to define the key concepts at play: conversational floor and turn-taking. Consider the following example of a dialog between a spoken dialog system (S) and a human user (U):

```
S: How may I help you?                          Miami?
U:                      I want to go to Miami.      Yes.
```

At a basic level, all we see is the transcription of words spoken by both participants over time. The conversational floor itself does not appear in the raw data. Yet, most people would agree that this dialog has four turns, that the floor first belongs to the system ("How may I help you?"), then to the user ("I want to go to Miami."), then to the system again ("Miami?"), and so on. While the idea that someone has the floor at a certain point in time in a conversation and that people take turns speaking make intuitive sense, they are in fact difficult to formalize. In fact, in their seminal paper, Sacks et al. [1974] never explicitly define what a turn is. Rather they describe turn-constructional units, of which turns are composed. But saying that a turn is a sequence of turn-constructional units presents the risk of a circular definition, as noted by [Clark, 1996, p. 324]. Sacks et al do not thus evaluate their model by comparing the turns it produces to some theoretical definition of what a turn is, but rather by ensuring that its output obeys a number of "grossly apparent facts", which act as empirical constraints on conversations. Perhaps, the two most fundamental of these facts are "Speaker-change recurs, or at least occurs", and "Overwhelmingly, one part talks at a time". Following this empirical approach, researchers who have focused on the analysis of recordings and transcripts of natural conversations, usually define the floor in a bottom up fashion. For instance, [Jaffe and Feldstein, 1970, p. 19] define possession of the floor entirely from the acoustic manifestation of a conversation (i.e. without regard to any linguistic or higher level aspect):

> The speaker who utters the first unilateral sound both initiates the conversation and gains possession of the floor. Having gained possession, a speaker maintains it until the first unilateral sounds by another speaker, at which time the latter gains possession of the floor.

This very objective definition is appropriate for the analysis of existing conversation, where in fact, anything except the sounds and the words has to be inferred by the researcher, and thus could be subject to ambiguity or disagreement. This is the definition we will use in our analyses of recorded human-human and human-computer dialogs.

On the other hand, this definition of the floor is not appropriate for a computational model of turn-taking *control*, i.e. for the model that an artificial agent engaged in a conversation would use to decide when to speak or be silent. While a bottom-up definition of the floor allows us to investigate *when* people start and stop speaking, it does not help us explain *why*. Why does the system stop, and the user start, speaking after "How may I help you?" The system proposed by Sacks et al relies on the hypothesis that participants in a conversation aim at minimizing gaps and overlaps between speech segments. However that criterion alone does not account for turn transitions, since the best way to minimize gaps and overlaps in a given dialog is to have one participant speak all the time and the other never.

To understand *why* users take turn, and ultimately build systems whose turn-taking behavior is, if not perfectly natural, at least optimal according to some criterion, we need to refer to a separate body of literature, that of computational linguistics. Computational linguists are more concerned with how people think rather than when they speak. They explain human conversational behavior in terms of beliefs, goals, intentions, and obligations [Cohen and Perrault, 1979, Allen and Perrault, 1980, Grosz and Sidner, 1986, Clark and Schaefer, 1989, Traum and Allen, 1994]. While these notions are often applied to high levels of processing (e.g. the user's goal in our example might be to book a flight to Miami), they are also relevant to turn-taking. For example, social obligations [Traum and Allen, 1994] explain why someone generally responds when being asked a question, whether or not they know the answer to the question, and whether or not they want to provide that answer. When asking a question, the speaker introduces an obligation for the addressee to respond. When an addressee raises her hand, opens her mouth and starts speaking, she manifests her intention to introduce new information or voice an opinion. Both obligations and intentions give rise to another definition of the floor, one that is subjective to each participant rather than objective as was Jaffe and Feldstein's. Subjectivity means that each participants in a dialog estimates at every point in time whether they or someone else have an obligation to speak, and whether another participant has the intention to speak. In this context, the only variable that a participant knows with certainty is whether they themselves have the intention to speak. Other participants' intentions are only known with a degree of uncertainty. In other words, each participant holds beliefs on other participants' intentions. This notion of the conversational floor based on beliefs, intentions, and obligations is one of the key aspects of the computational model of turn-taking we propose in chapter 6. It is also the definition that most spoken dialog systems implicitly use, even when their underlying turn-taking behavior is not based on any theory but rather on, potentially clever, engineering.

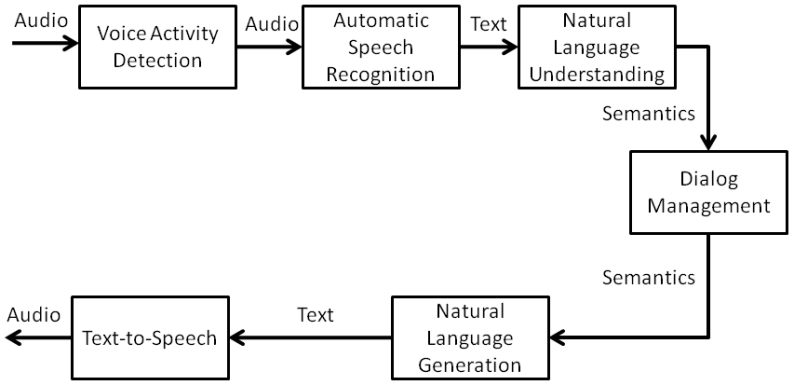


Figure 1.1: The typical pipeline architecture of a spoken dialog system.

1.3 Spoken Dialog Systems

Spoken dialog systems divide the complex task of conversing with the user into more specific subtasks handled by specialized components: voice activity detection, speech recognition, natural language understanding, dialog management, natural language generation, and speech synthesis. These components are usually organized in a pipeline as shown in Figure 1.1, where each component processes the result of the preceding one and sends its result to the next one. The following sections give a brief overview of each component and the typical issues they face.

1.3.1 Voice Activity Detection

Voice activity detection (VAD) is the problem of detecting in the incoming audio signal when the user speaks and when she does not. This apparently easy problem can in fact be extremely hard to solve accurately in noisy conditions and has been the focus of much research, particularly in the signal processing community. Virtually all approaches to VAD follow the same three step process:

1. Extract features from the audio signal
2. Classify small time frames (about 5-10 ms) as speech or non-speech
3. Smooth the classification decisions to identify speech and non-speech regions

The most straightforward feature is the energy of the signal. The classification decision is then based on a threshold on that energy. When the signal is louder than the threshold, the user is assume to be speaking. When the signal is quieter, the user is assumed to be silent. As long as the environment of the user is quiet, i.e. as long as the energy of noises other than user speech is significantly lower than that of user speech, this approach works well. However, when background noise, including potentially background speech, pollutes the audio signal, energy alone does not discriminate well between user speech and other noises, leading to many false positives (frames classified as speech when they should be silence). More advanced approaches have been explored [Ramírez et al., 2007, for a review of the state of the art], which have mostly focused on extracting robust features, while keeping the two other steps in the VAD process unchanged. Example of robust features are higher order statistics [Nemer et al., 2001, Cournapeau et al., 2006] and spectrum divergence measures [Marzinzik and Kollmeier, 2002].

1.3.2 Speech Recognition

The automatic speech recognition module (ASR) takes the speech audio data segmented by the VAD and generates its word-level transcription. In addition, the generated hypothesis is sometimes annotated at the word- or utterance-level with confidence scores. For example, given the waveform of the user uttering "I need to go to the airport" the output of the ASR should be `I NEED TO GO TO THE AIRPORT`.

ASR engines rely on three models [Huang et al., 2001], an acoustic model, which describes the mapping between audio data and phonemes, a lexicon, which describes the mapping between phoneme sequences and words, and a language model, which describes the possible (or likely) sequences of words in a language. The acoustic model needs to be trained on a corpus of transcribed utterances. The lexicon can be either trained as a set of letter-to-sound rules from a corpus of words and their pronunciation, or, more often, it can be written by hand. The language model can be either a hand-written grammar, or a statistical language model trained on a corpus of in-domain data. Most ASR engines are designed to process full utterances, where the definition of an utterance depends on the provided grammar or the corpus on which the statistical LM was built, but usually corresponds to a phrase or sentence. However, they usually perform recognition incrementally, as the user is speaking, and therefore can provide partial recognition results at any time, which, as we will see, can be used to inform turn-taking decisions.

1.3.3 Natural Language Understanding

The natural language understanding module (NLU) takes the sequence of words output by the ASR and generates a semantic representation of it. In the example above, the input to the NLU would be the sequence of words "I NEED TO GO TO THE AIRPORT", and the output could be the semantic frame `destination = ``THE AIRPORT"`.

NLU can be performed by providing a hand-written grammar that captures semantic relationships (either directly from the words, or via a syntactic analysis). Another approach to NLU is to train a statistical parser on a corpus of sentences annotated with their semantic representation. Most NLU modules assume that they are given a full utterance. Again the definition of an utterance varies and depends on the grammar or corpus on which the module was built. The traditional, text-based, approach to NLU [Earley, 1970, Charniak, 1997, see for example], which requires the input sentence to be fully parsable does not apply well to spoken dialog due to the large number of non-sentential fragments, as well as repairs and other disfluencies in sponetaneous speech. To cope with these irregularities, robust parsers have been proposed, which relax some of the constraints on the input sentence and can find the best parse for the utterance, even if it does not match perfectly the NLU's grammar [Ward, 1991, Lavie and Tomita, 1993]. Additionally, there has been some research on incremental parsers that are able to provide partial results before the utterance is completed [Wiren, 1992, Mori et al., 2001, Rose et al., 2002, ACL Workshop on Incremental Parsing, 2004, Kato et al., 2005].

1.3.4 Dialog Management

The dialog manager (DM) takes the semantic representation of the user input generated by the NLU and outputs the semantic representation of the system's response. In our example the input would be `destination= ``THE AIRPORT"` and the output could be `explicit_confirm:destination= ``THE AIRPORT"`, if the DM decided to respond to the user by asking them to explicitly confirm that their destination is indeed the airport.

While there are many approaches to dialog management [Larsson and Traum, 2000, Rich et al., 2002, Bohus and Rudnicky, 2003], the DM generally performs (at least) the following three tasks:

1. interpreting user input in the current dialog context
2. updating the dialog context based on user input

3. generating relevant system responses

To do so, the DM relies on some representation of the structure of dialog, which can range from simple finite-state machines as is the case, for example, in the VoiceXML standard widely used in industry, to complex, linguistically driven structures such as those proposed by Grosz and Sidner [1986] or Allen and Perrault [1980]. The DM also exploits knowledge about the domain and task at hand, which are usually provided by some back end module such as a database or an expert system.

1.3.5 Natural Language Generation

The natural language generation module (NLG) takes the semantic representation of the system response and outputs a natural language expression of it. In our example, the input would be `explicit_confim:destination= ``THE AIRPORT"`, and the output `Going to the airport. Is that right?`.

Simple and common approaches to NLG include canned text when there is little variation in system prompts and templates. In our example, the NLG might have used a template such as `Going to <destination>. Is that right?`. More advanced approaches have been proposed, either based on linguistic concepts such as discourse structure [Wilcock and Jokinen, 2003] or using statistical mapping between the semantic representation and the surface form [Oh and Rudnicky, 2000]. The NLG can optionally annotate the surface form with mark up tags destined to help speech synthesis using a speech synthesis mark up language such as SSML or JSAPI's. Such tags can indicate prosodic patterns such as rising or falling pitch, pauses, or emphasis.

1.3.6 Speech Synthesis

The speech synthesis, or text-to-speech module (TTS) takes the natural language output of the NLG (potentially augmented with mark up tags) and generates an audio waveform corresponding to its spoken version.

The simplest way to perform TTS, and also the one that leads to the highest naturalness, is to use pre-recorded prompts. As for canned-text NLG, this can be used when there is little variation in the system prompts so that they can all be covered by a voice talent. General speech synthesis allows the system to say any text, even potentially unplanned ones (which is necessary when the system retrieves variable information from, say, a web site). Common approaches to general-purpose TTS include concatenative synthesis, which

agglutinates segments of audio from a corpus of recorded utterances to produce new ones [Hunt and Black, 1996, Dutoit et al., 1996], and parametric synthesis, which generates the speech signal from a representation of the mapping from text or phonemes, to acoustics [Tokuda et al., 2000, Black, 2006]. The synthesized utterance is played back to the user through the output audio device.

1.4 Turn-Taking in Spoken Dialog Systems

1.4.1 End-of-Turn Detection

End-of-turn detection refers to the task of detecting when the user releases the floor after speaking an utterance. By far, the most common approach is to use the same rule that is used to segment the audio before sending it to the ASR. In other words, the system considers that the user releases the floor when the VAD detects a stretch of silence longer than a threshold. This approach has its merits, such as simplicity and a certain robustness. However, it faces two types of issues. On the one hand, the system can wrongly interrupt the user mid-utterance. We refer to this problem as a *cut-in*. On the other hand, the system can also fail to recognize the end of the user turn and remain silent when the user expects an answer, which will first slow down the interaction, and if prolonged, can lead to the user speaking again to re-establish the channel ("Hello?"), with the associated risk of misrecognitions and, again, confusion about who has the floor. We refer to the general problem of having too long delays before system responses as *latency*.

The system designer faces a trade-off between cut-ins and latency. By setting a short threshold, we can reduce latency at the cost of increasing the number of cut-ins, while a long threshold will reduce the number of cut-ins but increase latency. Figure 1.2 illustrates this trade-off in the CMU Let's Go bus information system [Raux et al., 2003, 2005, 2006, see also Chapter 3]. To plot this graph, we let the system pick a random threshold between 400 ms and 1200 ms at the beginning of each dialog (the threshold was kept constant throughout the dialog). We then binned dialogs according to the threshold that was selected (between 400 and 500 ms, between 500 and 600 ms, etc) and computed the cut-in rate for each bin. As expected, the graph shows that short thresholds (on the left side of the graph) result in higher cut-in rates, while long thresholds result in lower cut-in rates.

Cut-ins impact dialog performance in several ways. They confuse the user and can lead to situations where the user and the system keep interrupting each other, not knowing who has the floor. They also hurt speech recognition accuracy since they lead to incomplete user utterances that are usually poorly recognized. Hence, in the Let's Go system, we

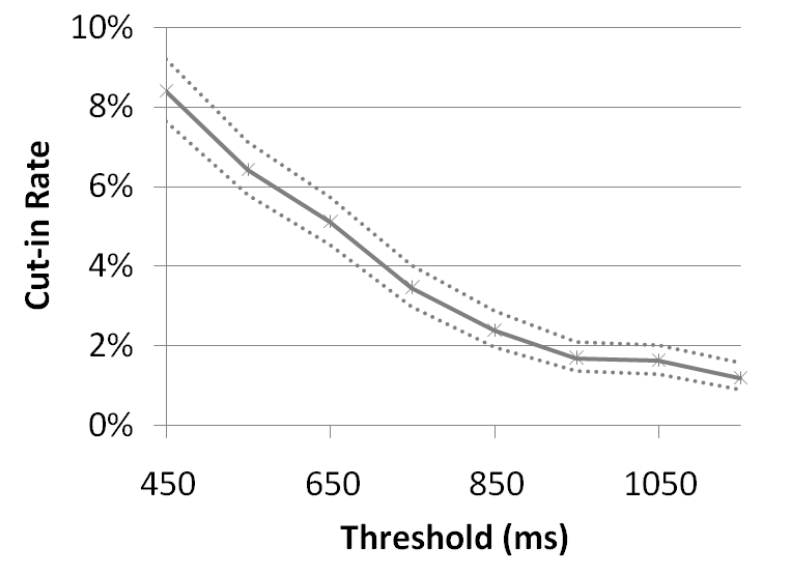


Figure 1.2: Latency / Cut-in trade-off.

found that ASR word error rate is significantly correlated with cut-in rate ($R^2 = 0.28$, $p < 0.0001$). The coefficient of the correlation equation is 3.6, indicating that on average, an increase of 1% of the cut-in rate corresponds to an increase of 3.6% of the word error rate.

On the other hand, excessive latency can also hurt user experience. In an experiment comparing dialogs with a dialog system with similar dialogs with a human operator, Ward et al. [2005] count system response time (i.e. latency) as one of the areas where an improvement could have a major impact on usability, noting in particular that:

> Responsiveness seemed to become relatively more important when the dialog departed from the desired path. In particular, swift exchanges were common during error recovery in the human-human dialogs but painfully absent during error recovery with the system.

In addition, they found that dialogs with the system tended to take more than 3 times as long as those with the human operator, with system latency accounting for 20% of the difference. Our own study of the Let's Go data, detailed in Section 3.5.4, confirms this, with the system being on average twice as slow to respond as human operators, and up to

18 times as slow after certain dialog acts (e.g. closings).

1.4.2 Barge-in Detection

Another important turn-taking phenomenon is *barge-in* or interruption. A barge-in happens when the user attempts to interrupt a system prompt. System designers typically take one of two approaches to handle barge-in: they either ignore it altogether, in which case the user is forced to listen to every system prompt in its entirety, or they base barge-in detection on the VAD alone, interrupting a system prompt whenever the VAD detects user speech. The issue with that second approach is that it exposes the system to detection errors by the VAD, in particular false positives, when the VAD misinterprets a noise (or background speech) as user speech. In such cases, the system interrupts itself when the user is actually listening, leading, at best, to significant confusion, and at worst to the user missing important information from the system. As for endpointing, there exists a trade-off between acting fast and acting accurately, as the system might be able to gather additional evidence in favor or against a barge-in (e.g. in the form of partial speech recognition results) by waiting after the VAD first detects speech. On the other hand, if the system fails to react in a timely fashion, the user might assume the system is not listening (or not accepting barge-ins) and cut their utterance short, leading to further turn-taking and recognition problems.

1.4.3 Other turn-taking phenomena

A number of other phenomena fall under the turn-taking category. For example, systems typically use a time out mechanism to reprompt the user if they fail to respond to a prompt within a certain time (or if the system failed to detect the user response). The issue of timeouts is quite straightforward in unimodal systems for which dialog is the only task. The timeout decision becomes more complex in cases where the user might be involved in other tasks during the dialog (e.g. driving, manipulating objects), which make the user less responsive but also act as evidence that the user has heard the system prompt and started to react to it, in which case a time out should not be triggered.

Finally, in certain tasks, such as in-car applications [Weng et al., 2004, Ko et al., 2005], the system might have to *intentionally* interrupt the user[1]. This can be crucial when dealing with time-sensitive (e.g. directions while driving) or critical (e.g. collision risk) information. This intentional interruption is to be distinguished from cut-ins, which happen when

[1]This is not to be confused with cut-ins described in section 1.4.1, which are unintentional

the system erroneously assumes that the user has released the floor. Knowing when to interrupt involves many levels of decision, some are in the turn-taking realm (i.e. finding an appropriate time in the user utterance to take the floor with minimum disruption and confusion), while others are at higher levels (e.g. how important is what the system wants to convey to the user compared to what the user is currently saying).

These phenomena, which involve some knowledge about the specific task at a high level and are not applicable to all dialogs, are beyond the scope of this thesis.

1.5 Principled Approaches to Optimizing Turn-Taking Behavior

Sacks et al. [1974] stated minimizing gaps and overlaps as one of the goals of a turn-taking system. Yet, the algorithms used for turn-taking in spoken dialog systems, relying on fixed rules involving very few, low-level, features do not generally make any attempt at optimizing their behavior. While there is still room for improving those features (e.g. more noise-robust voice activity detection), research on Conversation Analysis and psycholinguistics (see Chapter 2) tells us that the cues necessary to detect and anticipate turn-taking events come from all levels of dialog: acoustics, prosody, lexicon, syntax, semantics, discourse structure, and pragmatics. One difficulty of building machines that use those features for any kind of processing (turn-taking, discourse segmentation, emotion detection, etc) is that higher level features are usually harder to extract. Many questions need to be addressed even before one starts using that information. What level of semantics should be extracted? How to represent semantics and pragmatics? These issues are particularly daunting when dealing with free-form human-human conversation. Fortunately, spoken dialog systems, offer a constrained, task-oriented environment that not only makes the choices of representation easier, but even provides representations for higher levels of language. As we have seen, the speech recognizer, natural language understanding module, and dialog manager process user's input at different levels of representation. The dialog manager also contains a representation of discourse structure (i.e. the dialog state). The natural language generation and text-to-speech modules provide information at various levels of the system's contributions. However, besides the primary goal of dialog management, other components of dialog systems rarely exploit this wealth of knowledge.

In this dissertation, we describe a research program whose goal is to endow spoken dialog systems with a principled approach to turn-taking that allows to optimize the system's behavior based on data collected through interaction. Our approach relies on three components:

An architecture for spoken dialog systems where turn-taking and other low-level interactional phenomena are modeled explicitly

A model of turn-taking that represents the dialog's turn-taking state and controls the system's behavior

Optimization methods to allow the training and adaptation of the turn-taking model's parameters through interaction

Whenever possible, we will design the features and optimization procedures to allow unsupervised learning, so that the system can improve its behavior over time, without human intervention other than the dialogs themselves.

1.6 Thesis Statement

Incorporating different levels of knowledge using a data-driven decision model will improve the turn-taking behavior of spoken dialog systems. Specifically, turn-taking can be modeled as a finite-state decision process operating under uncertainty.

1.7 Contributions

This thesis extends the state of the art of spoken dialog systems technologies by providing:

1. an architecture for timing-aware spoken dialog systems (Chapter 4)
2. demonstration that dialog features can be used to improve turn-taking (Chapter 5)
3. an algorithm to optimize endpointing thresholds using dialog features (Chapter 5)
4. a decision theoretic model of turn-taking as a dynamic decision problem (Chapter 6)

Chapter 2

Related Work

2.1 Turn-Taking in Conversation

2.1.1 What is turn-taking?

In a conversation, most of the time one person speaks and the others don't. Participants "take turn" at the floor. While small gaps and overlaps between participants' speech are frequent they rarely last more than a few hundred milliseconds. This smooth interaction is one of the essential elements of spoken conversation, one that distinguishes it from other modes of communication such as monologues, formal speeches, and written and electronic mail.

The coordination between participants happens in a very natural way to anyone who has learned to speak, yet researchers have long failed to recognize it, let alone identify some of its mechanisms, signals and required skills.

In the 60s and early 70s, Harvey Sacks and his colleagues, created the field of Conversation Analysis (CA), defined as the study of how people interact with speech in different social settings (e.g. informal conversations, medical appointments, political interviews...). In particular, CA researchers have focused on turn-taking as one of the prominent features of spontaneous conversation. Based on an analysis of naturally occurring conversations, Sacks et al. [1974] established a set of constraints that they claim any model of turn-taking in conversation should obey. Namely, that such a model should be locally managed (i.e. only depends on the neighboring turns), party-administered (i.e. does not involve an external regulator), and interactionally controlled (i.e. managed through interaction between the participants). Another important aspect of turn-taking that needs to be accounted for

is that transition from one speaker to the next occur very frequently with neither gap nor overlap in speech, or at least not significant ones. Sacks et al propose a minimal model of turn-taking whose key components are a turn-constructional component, which defines what *is* a turn, and a turn-allocation component, which indicates *who* should speak next. A concept central to the turn-constructional component is that of Transition Relevance Places (TRPs). TRPs are points in an utterance where it would be *relevant* for another participant to take the floor.

2.1.2 TRPs and Turn Transitions

To explain the high number of turn transitions without gap (or even with overlap) that they observe, Sacks et al. advance the hypothesis that listeners are able to project TRPs before they occur. Many studies in CA following Sacks et al. [1974] aim at identifying features of TRPs and their projectability.

Sacks et al. [1974] consider syntactic constituents' boundaries as TRPs. This strict use of syntax is problematic considering that spoken language rarely has well-formed complex constituents due to disfluencies (see for example Levelt [1993] for a discussion of these issues). Recently, authors concerned with the quantitative analysis of turn-taking [Ford and Thompson, 1996, Furo, 2001] have given more operational definitions of syntactic TRPs. For instance, Ford and Thompson [1996] specify that they are "potential terminal boundaries for a recoverable *clause-so-far*". This is illustrated in the following example (from Ford and Thompson [1996]):

(2.1) V: And his knee was being worn/- okay/ wait./
It was bent/ that way/

In this utterance, the two syntactic completion points in "It was bent/ that way/" indicate that at these two points the whole units "It was bent" and "It was bent that way" can be considered complete. Often, a larger context must be taken into account to recover the clause-so-far. For instance, answers, which are often not complete syntactic constituents in spontaneous conversations, complete a clause *implied* by the question, and are therefore considered complete syntactic units from the point of view of turn-taking. Similarly, Reactive tokens such as backchannels (e.g. "okay", "mhm"), assessments (e.g. "really?") and repetitions (when the listener repeats all or part of the original speaker's utterance as an acknowledgment or confirmation) are often considered complete syntactic units although they are typically not well-formed syntactic constituents [Ford and Thompson, 1996, Sorjonen, 1996, Furo, 2001].

A second type of TRP features is prosody. In English, falling and rising pitches are usually considered as markers of intonation completion points for statements and questions, respectively [Chafe, 1992, Ford and Thompson, 1996, Furo, 2001]. In contrast, level pitch is considered a continuation marker, therefore not indicative of a TRP [Oreström, 1983, p. 62]. Another prosodic pattern is the lengthening, or drawling, of the final word or syllable of a turn [Duncan, 1972, Koiso et al., 1998], although these results are contradicted by Oreström [1983, p. 64]. These phenomena are sometimes accompanied by changes in voice quality such as creaks [Chafe, 1992]. Prosody as predictive of the end of turns has also been investigated by psycholinguists in perceptual experiments. For example, in an experiment on turn-taking in Dutch conversations, Wesseling and van Son [2005] asked their subjects to listen to recordings of natural conversations in two conditions: natural speech and synthesized speech, the latter preserving the prosody of the natural utterances but stripping them away of any lexical/semantic information. Subjects had to produce minimal vocal answers (backchannels) at times they felt were appropriate, which, the authors claim, would correspond to TRPs. The delay between each minimal response and the closest (manually labeled) turn boundary was then measured. Their results show that human listeners are consistently able to react very fast to, or even anticipate, turn boundaries both with full speech and with only prosodic information. This result is in partial contradiction with a previous study by Schaffer [1983], who found little consistency in how listeners use intonation to predict TRPs. In her experiment, subjects were listening to short extracts (”sentences, phrases, or single words”) from recorded natural conversations in again two similar conditions: normal recording and filtered. Based on what they heard, on one set of recordings, the subjects had to predict whether the next speaker in the original conversation (immediately after the extract) was the same as the one in the extract or a different one. Schaffer then analyzed whether groups of 20 to 31 listeners agreed in their judgements. She found only sparse evidence of agreement between the subjects. In particular, intonation alone (in the filtered condition) did not allow them to agree on points of speaker transition. Although part of the results might be explained by her experimental design (e.g. quality of the recordings, unnaturalness of the task, fact that speaker changes are often optional in natural conversation, etc), they do support the claim that intonation *alone* and *without context* is not a sufficient cue for turn-taking.

Semantic and pragmatic completion points correspond to points where the turn-so-far constitutes a complete meaningful utterance coherent within the conversational context. Not surprisingly, they have been found to strongly correlate with TRPs [Oreström, 1983, Furo, 2001]. However, all the authors addressing this point acknowledge the difficulty of establishing an operational definition of semantic completion. Hence, Oreström [1983, p. 57] writes:

> As there is no simple way to formalizing a semantic analysis of this conversational material, the interpretation will have to be made subjectively. Admittedly, this may be regarded as the weakest point in the whole analysis.

Furo [2001] uses a more specific definition of semantic completion points. They are placed after: floor-giving utterances (e.g. questions), complete propositions, and reactive tokens, provided they are not preceded by a floor-claiming utterance (such as "Here is what I told him."). However, there is still an issue with this criterion since it relies on syntax completion.

Finally, non-verbal aspects of face-to-face conversation are known to interact with linguistic signals for turn-taking. The most frequently observed such signal is that of speakers making eye contact with their listeners to indicate the end of their turn [Kendon, 1967, Goodwin, 1981]. On the other hand, gestures are generally found to be of little turn-taking significance [Oreström, 1983, p. 36], although Duncan [1972] did find that hand gestures were used by patients and doctors during interviews to cancel other turn-yielding signals. Beattie [1983] makes a distinction between simple movements, through which the speaker merely emphasizes the spoken message, and more complex gestures, which appear to accompany the planning of an upcoming utterance, the gesture generally preceding the verbal rendition of the utterance. While one can speculate about listeners using these gestures to predict upcoming speech (which would go along with Duncan's finding that gestures signal turn continuations rather than transitions), more studies are necessary in this area to expose the potential turn-taking functions of gestures.

2.1.3 Overlapping Speech

In addition to the signals announcing the end of a turn there are other turn-taking signals, in particular those that trigger backchannel feedback from the listener. Clancy et al. [1996] investigated the use of Reactive Tokens in English, Japanese, and Mandarin. They found that, in English and Mandarin, RTs are produced by the listener predominantly at syntactic completion points (resp. 78% and 88% of the RTs) whereas only 36% of RTs in Japanese conversation occur at such points. This, along with the fact that backchannels are much more frequent in Japanese (29.9% of all speaker changes are backchannels) than in English (15.9%) and Mandarin (4.7%), reflect the typical behavior in Japanese to produce backchannel after each noun phrase produced by the speaker (not only clauses). In her own work, Furo [2001] found a similar percentage of RTs occurring at syntactic completion points in English conversations (71.9%) but a much higher rate in Japanese conversations (83.7%). She also found that many RTs occur at noticeable pauses in both

English and Japanese (resp. 74.6% and 82.1%). Duncan and Fiske [1985] hypothesized that, in face-to-face interaction, the *speaker within-turn signal*, which triggers a backchannel from the listener, is composed of the completion of a syntactic clause and of a shift of the speaker's gaze towards the listener.

Ward and Tsukahara [2000] studied another potential backchannel trigger, again in English and Japanese. Their hypothesis was that regions of low pitch yield backchannel feedback. They built a predictive rule for each language and automatically predicted backchannels in two corpora, one consisting of 68 minutes of conversations between native speakers of English, and one of 80 minutes of conversations between Japanese speakers. Although the accuracy of their predictions is modest (18% for English, 34% for Japanese), it is above chance and shows that some of the backchannels can be explained by such a prosodic cue. Koiso et al. [1998] also used predictive models as an analysis tool and trained decision trees to predict backchannel occurrence in Japanese conversations. The recordings were automatically split into "interpausal units" (stretches of speech separated by pauses of 100 ms or more). For each unit, a number of syntactic and (discretized) prosodic features was semi-automatically extracted. The authors first studied the correlation between each feature and the occurrence of a backchannel in the pause following the unit and then built a decision tree using all the features to predict the backchannels. As a result, they are able to predict backchannel occurrence with a high accuracy (88.6%). They found that intonational features (the type of final contour of the unit) are the strongest signals for the occurrence of backchannels. In contrast, the most prominant features inhibiting the occurrence of backchannels are syntactic (i.e. certain types of words, such as adverbs, tend to "block" backchannels). In her analysis of videotaped conversations between strangers, Denny [1985] built logistic regression models to predict the occurrence of backchannels during pauses of 65 ms or more. Although she does not provide an estimate of the quality of the prediction, she did find that the most useful (hand-annotated) features were grammatical completion, speaker gaze, intonation, pause length, and type of preceding utterance (questions are less likely to be followed by backchannels than statements). Cathcart et al. [2003] attempted to predict backchannel feedback in the HCRC Map Task Corpus of goal-directed conversations using low-level, easy to extract features. By combining pause duration information with a trigram language model built on symbols representing Parts-of-Speech and pauses from the main speaker, as well as backchannels from the listener, they were able to achieve precisions in the range of 25%-30% and recalls of 35%-60%.

As we see, the literature on cues triggering backchannel feedback indicates that somewhat similar features are used for backchannel and turn transitions. One difference is that backchannel feedback is always optional (particularly in English), whereas turn transi-

tions are more systematic. Indeed, one can imagine a completely functional conversation without any backchannel, but not without turn transitions. Therefore, it is difficult to evaluate the acuracy of backchannel predictions, since they attempt to detect places where a backchannel feedback *could*, but does not have to, be produced. As for turn transitions, it seems that one weakness of all the analyses described above is that, to a large extent, they ignore semantic and pragmatic information, which might be of prime importance in the decision to produce backchannel feedback (e.g. the fact that a speaker sounds sad or concerned might lead a caring listener to produce backchannels, as a form of support).

2.2 Turn-Taking Models for Spoken Dialog Systems

2.2.1 Supporting and related work

Incremental Language Understanding

Research in psycholinguistics has established that humans process utterances incrementally [Tyler and Marlsen-Wilson, 1977, Altmann and Steedman, 1988, Kamide et al., 2003]. That is to say, when we hear an utterance, at each point during the utterance, we hold a (potentially underspecified) semantic representation of this utterance. Both in order to match human language processing and to allow interactive natural language-based applications, incremental parsers have been proposed and implemented by computational linguists [Wiren, 1992, Mori et al., 2001, Rose et al., 2002, ACL Workshop on Incremental Parsing, 2004, Kato et al., 2005]. Among those, a number have specifically targetted spoken dialog systems.

For example, Stoness et al. [2004] describe a continuous understanding module for a dialog system which takes into account context information (e.g. reference resolution) to guide its parser dynamically. They use a Mediator module which exploits high level information (e.g. from the reference resolution module of a dialog system) to directly modify the chart of a chart parser, changing the probability of certain chart entries or adding new chart entries. They evaluated their approach on a corpus of human-human dialogs and found that it brought some, albeit quite small, improvement over a non-incremental parser, in terms of understanding accuracy and efficiency.

The incremental parser developped by Nakano et al [Nakano et al., 1999b], uses a unification-based grammar and combines NLU and discourse processing in an algorithm called Incremental Significant-Utterance-Sequence Search (ISSS). For each new incoming word from the ASR, the parser maintains a number of candidate contexts (i.e. parsing stack

and beliefs about the user's intention), each associated with a priority level. Priorities are heuristically set so as to be higher for "significant utterances", i.e. phrases or clauses that correspond to a fully formed dialog act, and also favor longer constituents (as opposed to concatenation of shorter ones). This parser is part of the WIT spoken dialog system toolkit (see below).

Another related work is Wang [2003], which describes an integrated recognition and parsing engine. In this approach, the speech recognizer uses a combination of Probabilistic Context-Free Grammars (PCFG) and N-grams language models to capture the semantic structure of the input (through the PCFG), while keeping the flexibility of N-grams to model the language of complex semantic classes (e.g. "email subject"). Decoding is then performed frame by frame, as in standard speech recognition, and the current hypotheses, including their semantic PCFG classes, can be obtained at any frame. The method was evaluated in MiPad, a personal assistant for mobile devices that allows the user to manage contacts, emails and schedules. The results show no difference between the synchronous (i.e. incremental) NLU version and a standard "turn-based" version in terms of task completion rate or time. However, subjects using the incremental NLU version tended to produce longer and more complex utterances since the system showed its current understanding while the user was speaking.

Real-Time Control of Mobile Robots

The incremental processing work described in the previous section is rooted in human language processing and psycholinguistics. Another way to approach flexible turn-taking is as an instance of real-time reactive behavior by an artificial agent. This relates to research in Artificial Intelligence and mobile robotics on how autonomous mobile robots can navigate in the real world and perform their tasks. A common principle used to build architectures for such agents is to use multiple layers [Brooks, 1985, Muller, 1996, Bonasso et al., 1997, Hassan et al., 2000]. In such architectures, the components in charge of long-term planning (such as those required to explore a building) and those in charge of immediate reactive behavior (such as those required to avoid obstacles) operate asynchronously and for the most part independently. They communicate usually by passing messages. On the one hand, the reactive component informs the planning component of events arising in the real world that might trigger replanning. On the other hand, the planning component informs the reactive component of its plan of action so that the reactive component can (attempt to) take the prescribed actions. This separation into layers can be ported to spoken dialog, where the planning component is the dialog manager and the reactive component (which typically does not exist in standard dialog systems) is in charge of executing the actions

(e.g. saying the utterances) decided by the dialog manager while monitoring real-world events (e.g. barge-in, backchannel) that might modify the dialog manager's model of the world.

2.2.2 The Ymir Architecture

Possibly the most detailed work on reproducing human turn-taking behavior in artificial conversational agents is that of Thorisson [Thorisson, 1996, 2002]. The core of this work is Ymir [Thorisson, 1999], an architecture for conversational humanoids that integrates many aspects of multimodal face-to-face interaction (e.g. hand gestures, gaze, backchannels, but also higher level aspects such as discourse planning) in a unified framework. Although the architecture is presumably task- and domain-independent, all of the work in turn-taking has been done with Gandalf, an embodied agent acting as a guide to the solar system. As in other systems, the architecture contains three major sets of agents: perceptors (and multimodal integrators, which combine information from unimodal perceptors), deciders, which plan the interaction, and behaviors, which encode the realization of communicative actions as motor sequences. In addition, Thorisson divides mental processes involved in conversation in three layers, which affect all three types of modules (perceptors, deciders, and behaviors). These layers correspond to different levels of priority and reactivity and are, from the highest priority to the lowest:

the Reactive Layer which captures the fastest (and simplest) behaviors such as broad-stroke functional analysis and reactive behaviors (e.g. backchannels). Decisions in this layer are made at a frequency of 2-10 per second.

the Process Control Layer which captures domain-independent dialog structure interpretation and control mechanisms. Such decisions happen 1-2 times per second.

the Content Layer which contains actual linguistic and higher level mechanisms used for content interpretation and presentation. Decisions at this level happen 1 or less times per second.

The argument is that, given the constraints of real-time processing imposed by face-to-face interaction, an agent should first analyze the other participant's behavior in terms of broad functionality (e.g. is a gesture communicative or non-communicative?), then in terms of dialog control (e.g. turn-taking), and only finally (i.e. "when time allows") in terms of linguistic and discourse content. In practice, deciders and multimodal integrators are coded as sets of rules that get evaluated at a rate depending on their layer. For example,

one rule for turn-taking lying in the Reactive Layer is (in LISP-style, reproduced from Thorisson [2002]):

```
START STATE: Other-Has-Turn
END STATE  : I-Take-Turn
CONDITION  : (AND  (Time-Since 'Other-is-presenting > 50 msec)
                   (Other-produced-complete-utterance = T)
                   (Other-is-giving-turn = T)
                   (Other-is-taking-turn = F))
```

This rule, evaluated 2-10 times per second, specifies that the system must take the turn after a 50 ms pause following a user utterance, provided the utterance is complete and turn-yielding. Other examples of rules in the Reactive Layer aim at showing that the system is listening (by modifying Gandalf's facial expression) or at looking puzzled during an awkward pause. Rules in the Process Control Layer concern higher level behavior such as turning to the user when the system speaks. Gandalf/Ymir was evaluated in three ways: comparison with human behavior, reusability of the framework, and user study [Thorisson, 1996]. First, from a theoretical point of view, Thorisson compared the response times of Ymir's various perception, decision, and action loops with those of humans and found that they were roughly comparable. This evaluation is however highly dependent on the performance of the hardware and the software used and is probably no longer relevant ten years later. Second, Thorisson showed that his framework is general by using it in two very different systems: Gandalf and Puff the LEGO Magic Dragon, a character in a video game-like environment. More interestingly, he conducted a user study in which 12 subjects interacted with three versions of Gandalf: one with just content (task) related behavior (CONT), one with additional turn-taking abilities (ENV), and one with emotional expressions (e.g. looking confused) but no flexible turn-taking (EMO). In the follow-up questionnaires, subjects rated the ENV system higher than the other two in terms of smoothness of interaction and life-likedness. More surprisingly, they also rated the system's ability to understand and produce natural language significantly higher in the ENV condition than in the other two. One explanation for this would be that the answer to the natural language questions reflected more the overall satisfaction of the subjects than the specific understanding and generation ability of Gandalf.

2.2.3 The TRIPS Architecture

The TRIPS spoken dialog architecture [Ferguson and Allen, 1998], developped at the University of Rochester, is an intergrated system for conducting collaborative problem-solving

dialogs. It has been used to develop a number of dialog systems over almost a decade on tasks such as emergency response and evacuation planning [Allen et al., 2000]. While its initial implementation handled turn-taking in the standard rigid way, a later version featured a new core architecture specifically allowing incremental interpretation and generation [Allen et al., 2001]. This new architecture is divided in three components: interpretation, generation and behavior, the latter representing all high level planning. An important feature of TRIPS is the separation of discourse- from task-related components. Discourse is captured by a Discourse Context (DC), which contains typical elements such as the past history of user and system utterances and the set of current salient entities (for reference resolution), but also discourse level obligations (e.g. the obligation to address a question asked by the user), and current turn status. All the components of the architecture run incrementally and asynchronously, which allows for flexible turn-taking behavior. For example, when there is an obligation to respond to the user in the DC, the "normal" behavior is to get the answer to the user's question and produce it. However, the generation module might also decide to provide an acknowledgment, possibly in the form of a backchannel, without waiting for the task-reasoning modules' complete answer. Unfortunately, while Allen et al. [2001] do provide a number of examples of the phenomena that this architecture aims at capturing, there are, to my knowledge, no published studies of a full spoken dialog system based on it exploiting its turn-taking capabilities. There is, however, recent work [Allen et al., 2005] using a TRIPS-based multimodal system that takes speech input to manipulate graphical objects on the screen, but responses from the system are only graphical.

The WIT Architecture

The WIT spoken dialog system toolkit was developped at NTT Corporation by Nakano and his colleagues [Nakano et al., 1999a, 2000]. This toolkit provides four main modules. First, a grammar-based speech recognizer outputs word hypotheses as they are identified as being part of the best path in the grammar. Second, the core of the architecture is the incremental natural language understanding module described in section 2.2.1. The third component of the architecture is an incremental natural language generation module described in detail in Dohsaka and Shimazu [1997], which takes into account user responses (including backchannels) to system utterances to dynamically update output planning. Finally, the last module concerns speech output and consists of pre-recorded phrases that are concatenated together according to the NLG output plan. While systems built on the WIT architecture have been used in further research studies (e.g. the work on utterance boundary detection in Sato et al. [2002]) and videos demonstrating such systems have been made public, no formal evaluation of this system has been published.

2.2.4 Other work

Lemon et al. [2003] proposed a multi-layered architecture for dialog systems with the goal of separating the planning aspect of dialog from its reactive aspect. The top layer keeps track of the context of the dialog and plans responses to user requests and system-initiated topic switches. The bottom layer is in charge of maintaining the communication channel. This involves a variety of tasks such as the generation of outputs scheduled by the top layer, targeted help after non-understandings, and turn-taking (e.g. interrupting the user). None of the actions taken at the bottom layer require information about the context of the dialog beyond the last system and user utterances. The two layers operate asynchronously and communicate through specific data structures. For example, the Output Agenda is a prioritized list of the outputs planned by the top layer that is accessed by the bottom layer to decide what to generate next. Again, while the authors give examples of behaviors made possible by the architecture and mention a system built on top of it, no formal evaluation results are provided.

Hulstijn and Vreeswijk [2003] use turn-taking as a test case for agent-based programming. Thus their goal is neither to validate a linguistic theory nor to build a conversational agent, but rather to implement the SSJ model of Sacks et al. [1974] as multi-agent software. In particular, the simulated multi-party dialog that they generate have no linguistic content, participants selecting utterances at random from a fixed set when they decide to speak. However, Kronild [2006] describes the turn manager of an actual artificial conversational agent partly inspired by this work. As Hulstijn and Vreeswijk [2003], it follows the SSJ model closely and allows for multi-party conversation. This model relies on complex state machines called Harel statecharts [Harel, 1987]. Unfortunately, while this turn-taking model was built in conjunction with a dialog manager for multi-party conversation [Kronlid, 2008], this works remains essentially theoretic and does not mention any empirical evaluation. Our own model, described in Chapter 6, also relies on a finite-state machine, albeit a simpler one.

Aist [1998] describes a turn-taking architecture for the Reading Tutor of CMU's Project LISTEN, a system that listens to children reading aloud and provides help and feedback accordingly. Although the Reading Tutor differs from task-based spoken dialog systems in that the student spoken input is limited to reading (correctly or not) sentences displayed on the screen, thus eliminating the problem of natural language understanding, it does involve a multimodal "dialog" in that, in addition to reading the sentence, the student can click on certain GUI elements (e.g. to request help). On the other side of the interaction, the system uses speech and graphics to provide instructions, feedback, and help to the student. The tutorial actions that the system can take [Aist and Mostow, 1997] include backchanneling

when the student is correctly reading the sentence. In order to accomodate the multimodality and allow for time-sensitive actions, the system uses an event-based model, where each event (examples of events include ”user starts speaking”, ”user clicks on the Back button” and ”tutor stops speaking”) is timestamped and certain events trigger transitions in a finite state discourse model. Based on the incoming events and the current state of the interaction, a set of turn-taking rules is used to decide, 5 times per second, whether the system should start/stop speaking or produce a backchannel. Unfortunately, little quantitative evaluation of the turn-taking mechanism of the system has been published.

2.3 Summary

In this chapter, we discuss how decades of research on human-human conversation, both by conversation analysts and psycholinguists, first uncovered the turn-taking process and then revealed its complexity. In particular, both the analysis of naturally occurring conversations and perceptual experiments have shown that humans resort to a wide range of features in order to make timely turn-taking decisions.

We present a number of efforts to apply these findings to spoken dialog systems through flexible architectures that rely mostly on incremental natural language processing, and on robotic principles of real-time, event-driven, control.

Chapter 3

Research on a Deployed SDS: the CMU Let's Go Bus Information System

3.1 Summary

In this chapter, we present the task and system that we used for all the data collections and evaluations described in the following chapters. We begin by discussing the advantages and challenges of doing research on publicly deployed systems in section 3.2, and explain our decision to base our research on Let's Go, a publicly deployed bus information system for the city of Pittsburgh. Section 3.3 provides an overview of the system and of its speech understanding, dialog management, and speech generation components. In section 3.4, we describe the results of Let's Go's public deployment in terms of call traffic and task completion. Finally, in section 3.5, we analyze in more details the turn-taking behavior of the baseline system, which relied on standard voice activity detection and threshold-based endpointing techniques. We find that turn-taking failures are frequent and can lead to dialog breakdowns. Further, we compare temporal aspects of turn-taking between successful Let's Go dialogs and natural human-human dialogs on the same task. Results show that the system is significantly less efficient at taking turns than a human operator and that part of this difference comes from the inability of the system to leverage information such as dialog state when making turn-taking decisions. Ignoring such information leads to unnatural turn transition pauses, which in turns affect user's own turn-taking behavior.

3.2 Research on Deployed Spoken Dialog Systems

Until recently, virtually all research on spoken dialog systems was conducted in the lab. Even today, most academic work relies on systems (either fully automated or relying on some form of the Wizard-of-Oz paradigm) specially created for the purpose of a study. Experiments rely on recruited subjects that are given tasks to accomplish with the system, and usually some form of compensation for their participation. In the past decade, the emergence and spread of telephony based systems used by thousands, if not millions, of users have given birth to a new kind of study, which relies heavily on the analysis of large amounts of data collected from the logs of those systems Raux et al. [2006], Damnati et al. [2007], Bacchiani et al. [2008]. Here we describe the pros and cons of both approaches to spoken dialog systems research and explain our choices for the evaluation of the work described in the following chapters.

The first question one has to consider is whether a system matching the researcher's topic of study already exists. For research that explores new applications of the technology (e.g. human-robot interaction), the only solution is for the researcher to create a new system and, most of the time, to focus on controlled experiments in the lab. However, many research questions span a wide variety of tasks. For those, whether to exploit a system deployed among real users or to conduct controlled experiments in the lab is a genuine design question. One of the main advantages of deployed system is that they provide large amounts of collected data on which to train and evaluate models. As anyone involved in user studies is well aware of, collecting large amounts of data through controlled experiments is a very costly, time-consuming, enterprise. A deployed system generally provides a constant stream of data without the need for recruiting users. In addition, users of a deployed system are genuinely trying to achieve their goal, whereas paid subjects' motivation is external to the system or task. As a consequence, as found by Ai et al. [2007], dialogs in lab settings differ from those with deployed systems along various dimensions such as speaking rate, help requests, and barge-in. Evaluating dialog systems in their "natural" setting, with real users, is thus a major benefit of research on deployed systems. On the other hand, lab studies allow for far more control over the various variables that affect the performance of a system, such as task difficulty, speech recognition accuracy, etc. Such experiments might be able to capture subtle differences that would be otherwise masked by other dominant factors in a deployed system.

Because turn-taking is a universal conversational phenomenon that occurs in all human-computer spoken interaction, not only advanced research systems, but also simple form-filling systems such as those that are already commercially available today, we decided to focus on a relatively standard, but deployed, system. Specifically, the research presented

in this thesis is based on Let's Go, a public, telephone-based dialog system that we fully designed and implemented, and deployed in partnership with the Port Authority of Allegheny County, which operates buses in the Pittsburgh area. We believe that this choice allowed us to:

1. Gather enough data to build data-driven models of turn-taking
2. Assess the proposed approaches on a wide variety of users and conditions
3. Confirm the potential of advanced turn-taking behavior on current systems

with the caveat that the task-at-hand (providing bus schedules) might not be as impacted by improved turn-taking as would be a more artificial task purposefully designed to emphasize turn-taking (e.g. one where the user needs to provide long lists of items, or one of a real time nature such that the system might have to sometimes interrupt the user). Ultimately, however, the results presented in chapters 5 and 6, indicate that even simple systems can benefit significantly from this work.

3.3 Overview of the Let's Go System

Let's Go [Raux et al., 2003, 2005, 2006] is an automated telephone-based service that provides access to bus schedules for certain routes in the Port Authority of Allegheny County network, covering parts of the Pittsburgh metropolitan area. Since March 2005, users who call the Port Authority outside of business hours (i.e. between 7pm and 7am on weekdays and 4:30pm and 8am on weekends), are given the possibility to use Let's Go. This results in an average of 40 calls per night on weekdays and about 80-100 calls per night on weekends handled by the system. Software-wise, Let's Go is built on the Olympus dialog system framework Bohus et al. [2007]. Figure 3.1 shows an example of a typical dialog with Let's Go. The following sections describe Let's Go's components.

3.3.1 Speech Recognition and Understanding

Two engines using gender-specific acoustic models and context-dependent language models run in parallel. Originally, the Sphinx-II engine Huang et al. [1992] was used, which was subsequently replaced by the newer PocketSphinx engine Huggins-Daines et al. [2006] in April 2007. The top (1-best) hypotheses of both engines are parsed and the system selects one of the hypotheses based on the dialog-level confidence annotation performed by

S: Welcome to the bus information system. This conversation will be recorded for technical research purposes. What can I do for you?
U: 54C
S: The 54C . Did I get that right?
U: YES
S: Alright. Where are you leaving from?
U: OAKLAND
...
S: Leaving at 8 p.m. Is this correct?
U: YES
S: Right. Hold on. Let me check that for you. There is a 54C leaving FIFTH AVENUE AT BIGELOW at 8:19 p.m. It will arrive at EAST CARSON STREET AT TWENTIETH at 8:28 p.m. To get more information about buses related to this trip, you can say, when is the next bus, or, when is the previous bus. To ask about a different trip, you can say, start a new query. If you are finished, you can say goodbye.
U: WHEN IS THE PREVIOUS BUS
S: Okay. There is a 54C leaving FIFTH AVENUE AT BIGELOW at 8 p.m.
...

Figure 3.1: Excerpt from a dialog with the system. (U: user turns, S: system turns)

Helios (see below). As of January 2007, the lexicon, in which each bus stop name appears as a single multi-word entry, contains 7913 words. The language model was initially trained on data collected in experiments predating the opening of Let's Go to the general public. This resulted in a significant mismatch between the model and the actual users' utterances, and thus to poor recognition accuracy. In June 2006, we retrained the language and acoustic models on a corpus of 2000 calls to the public system that were manually transcribed[1]. For the language model, the data was used to train both the general parts of each dialog state's LM, as well as distributions within word classes such as place names, neighborhoods, and bus routes.

Figure 3.2 shows hand labeled task success as a function of word error rate (WER, computed as the ratio of the number of words inserted, deleted and substituted by the speech recognizer divided by the total number of words actually spoken by the user), which is a standard measure of speech recognition accuracy. Except for the left-most point where too little data was available, it can be noted that there is a strong linear correlation

[1]Many thanks to David Huggins-Daines for performing the acoustic training/adaptation

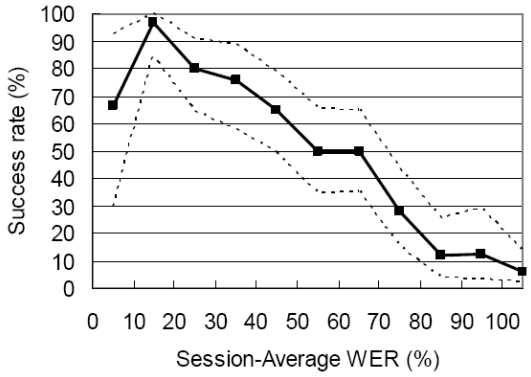


Figure 3.2: Relationship between word error rate and task success in the Let's Go system.

between the two ($R^2 = 0.97$ for WER over 10%). This was confirmed when we retrained the models in 2006, bringing about a significant reduction of the word error rate as shown on Figure 3.3, from about 60% to about 35%. At the same time, dialog completion rate (an automatic estimate of task success, defined in section 3.4.2) rose from 45% to 75% (see Figure 3.5(b)).

For natural language understanding, Let's Go uses Phoenix, a robust semantic parser using hand-written context-free grammars. The grammar is designed so as to directly capture relevant concepts and their value, without requiring an intermediate syntactic parse. It was initially written based on intuition and initial wizard-of-oz and in-lab experiments, and continuously refined as dialogs with general users were transcribed and analyzed.

The last part of the understanding process is confidence annotation, which is performed by the Helios confidence annotator developed by [Bohus and Rudnicky, 2002]. Helios relies on a logistic regression model trained on a large number of features, including recognition score, parse coverage, dialog state, etc. The model used in Let's Go was trained on 9163 dialog turns collected with the Let's Go system during the first three weeks of public use in March 2005.

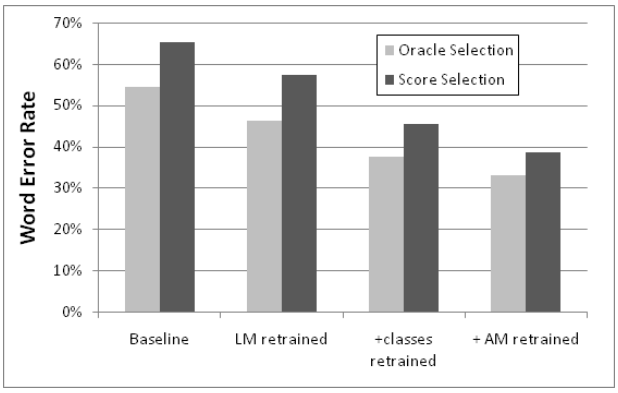


Figure 3.3: Effect of language and acoustic model retraining on word error rate. As explained in section 3.3.1, two gender-specific engines were used. The dark bars represent WER obtained when selecting for each user utterance the recognizer with the highest (automatically computed) recognition score, while the light bars represent WER obtained when selecting the recognizer with the lowest WER (oracle selection). The latter is a lower bound of the performance obtainable by selecting one recognizer for each utterance. At runtime, the selection is based on Helios confidence (see text), which, in addition to recognition score, uses information from the parser and dialog state. Therefore, runtime performance typically lies between the two bounds given here.

3.3.2 Dialog Management

Task Specification

Let's Go uses a plan-based dialog manager using the RavenClaw Bohus and Rudnicky [2003] framework. RavenClaw provides a generic dialog manager to which the system designer must provide a task specification in the form of a tree. The tree captures dialog structure by decomposing conversational goals into subgoals, from high-level tasks such as "getting the query specifications from the user" to atomic dialog actions such as "asking where the user is leaving from". While by default, RavenClaw performs a depth first left-to-right traversal of the tree, the task specification contains preconditions and triggers that allow for more complex dialog flows. Figure 3.4 shows the dialog tree for the Let's Go system as of March 2008.

The core structure of the dialog is as follows. The system first greets the user, then acquires the details of the user query, provides results, and allows the user to ask follow-up requests (e.g. asking the bus immediately following the given result). The user query has three or four slots: departure place, arrival place, travel time, and optionally, bus route. In order to help dialogs with understanding problems recover gracefully, we implemented a specific sub-dialog that is triggered after 2 unsuccessful attempts at getting the departure place from the user. In this dialog, the system first asks the neighborhood and then narrows down to the specific stop, leaving the user the option to say "I don't know" and let the system pick a common stop in the chosen neighborhood.

Opening prompt

While the core of the dialog is system-directed, we experimented with three types of opening prompts (see Raux et al. [2006]):

1. Which bus number or departure place do you want information for?
2. What bus information are you looking for?
3. What can I do for you?

In version 1, the system only recognized bus numbers and places at this point in the dialog, whereas in version 2 and 3, the system could understand more general utterances such as When is the next bus from CMU to downtown?. If the system failed to understand anything on the users first utterance, it gave a help message with examples of appropriate

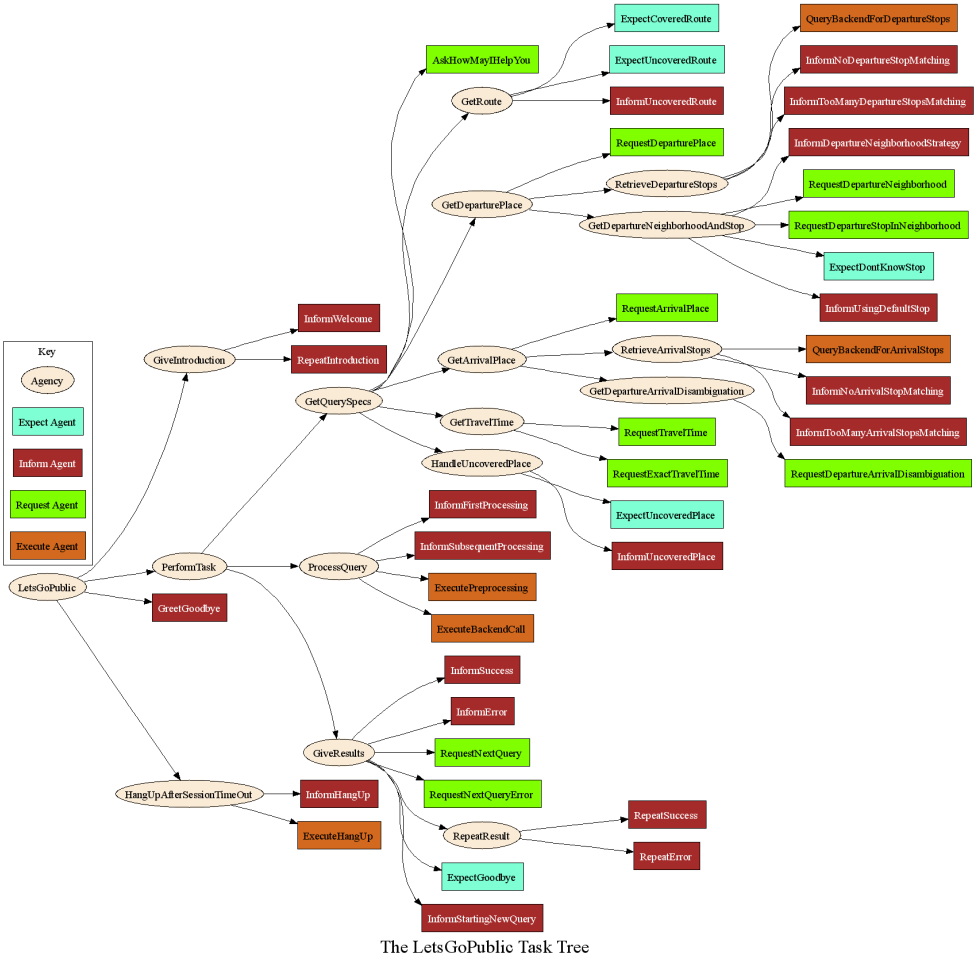


Figure 3.4: The RavenClaw task tree for the Let's Go spoken dialog system (March 2008). The tree is rotated 90 degrees, with the root on the left and leaves to the right. Left-to-right traversal becomes top-to-bottom in this layout.

System version	Nb of dialogs	Call duration (ms)	Non-underst. rate	Dialog completion rate	Average nb of user turns
1	1063	1678	32.2%	54.9%	17.9
2	1006	1750	28.9%	52.4%	17.5
3	999	2828	52.0%	51.5%	18.2

Table 3.1: Impact of initial prompts initiative style on user behavior and system performance

utterances (the examples were different for version 1 vs 2 an 3). Table 3.1 shows various metrics of user behavior and system performance in the three conditions.

Interestingly, although prompts 2 and 3 were designed to be similar, except for their specific wording, user behavior following prompt 2 was much more similar to that following prompt 1. Apparently, many users understood prompt 2 as asking them "Which bus do you want information for?", leading them to respond simply with route numbers, quite similarly to what people responded to prompt 1. In contrast, prompt 3 led to significantly longer user utterances, as well as significantly more non-understandings. However, the differences in both task completion rate and overall number of turns per dialog were small and not statistically significant. Therefore we decided to keep the open prompt 3 as the standard opening for Let's Go, so that we could collect more data in this condition and improve recognition accuracy.

Grounding strategies

Another important aspect of dialog management is error handling. RavenClaw provides a principled approach to confirmation, as well as a set of domain-independent dialog strategies for handling non-understandings. Grounding behavior does not need to be represented in the task specification tree (as can be seen in Figure 3.4) but rather is performed as a background task by the dialog engine. However, the dialog designer can specify which strategies to use for confirmations (e.g. alway explicitly confirm concepts or mix implicit and explicit confirmations) and non-understanding recovery (e.g. see Table 3.2 for example of non-understanding recovery strategies).

For confirmation, except for the final experiment described in Chapter 6, Let's Go has been relying on systematic explicit confirmation of every concept. We decided to

Strategy	Old set	New set
General help on how to use the system	X*	X
Local help + context-dependent examples	X*	
Context-dependent examples	X*	X
General help + local help + c.-d. examples	X*	
Give up question and go on with dialog	X	X
Repeat question	X*	X
Ask user to repeat what they said	X	X
Ask user to rephrase what they said	X	X
Ask user to go to a quiet place		X
Ask user to speak louder		X
Ask user to use short utterances		X
Offer to start over		X
Give up dialog and hang up		X

Table 3.2: Non-understanding recovery strategies in the old and new version of Let's Go! (*: the prompt for this strategy was preceded by a notification prompt)

take this rather conservative approach in order to limit the confusion on the user and the need to open the initiative (to allow corrections of previous concepts), which could lead to degraded performance.

For non-understandings, the initial set of strategies was designed based on our intuition and our experience with research spoken dialog systems. This original set is described in Table 3.2. In early 2006, having learned a lot from almost a year of experience with a real-world system, we modified the set of non-understanding recovery strategies (see Table 3.2). The modifications were of three types: rewording of system prompts, removal of ineffective strategies, and addition of new strategies.

Our experience with Let's Go suggested that long prompts were not well received by the users and were mostly ineffective. Consequently, we removed non-critical informational content from prompts, shortened them, and made them as specific as possible. For example, many prompts started with a notification that a non-understanding had occurred ("Sorry, I didn't catch that.").

During such prompts, users would frequently barge in on the system right after the notification and thus not hear the following utterance, which contained help or examples of

expected user utterances. We therefore decided to eliminate the notification prompt so that the user could hear the more informative specific content of each prompt. We also removed generic help prompts, which explain what the system is trying to do at a given point, since they didn't appear to help much. However, we kept the help prompts giving example utterances (e.g. "For example, you can say When is the next 28X going to the airport."), which were more often picked up by users and were thus more effective. Finally, we added more specific strategies, aiming at dealing with problems like noisy environments, too loud or too long utterances, etc. The idea was that such pinpointed strategies, if used at the right time, would be more effective in addressing the issues hurting communication between the user and the system. Currently we use simple heuristics to trigger each of these strategies, based for example on the length of the last user utterance or a simple audio clipping detection algorithm. To evaluate the impact of these changes, we manually labeled the action following each non-understanding as successful when the next user utterance was correctly understood by the system or failed when the next user utterance led to another non-understanding or to a misunderstanding. We labeled three days of data before the modifications took place and three days after. We used the same days of the week (three weeks apart) to mitigate the effect of daily variations in performance. The results indicate that the modifications significantly improved the success rate of non-understanding recovery strategies, from 19.8% to 24.1% ($p < 0.01$). The overall dialog completion rate also went up from 49.7% to 55.2% although this result is only a trend ($p < 0.1$).

3.3.3 Speech Generation and Synthesis

Let's Go uses Rosetta, a template-based natural language generation module. Hand-written templates, which can be as advanced as full programmatic functions (in Perl), are used to convert a semantic frame generated by the dialog manager into an English sentence. This sentence is then passed to Cepstral's Swift synthesizer, for which a limited-domain voice was built. The corpus on which the Let's Go voice was trained contains 1702 in-domain utterances (generated by the system's natural language generator, using place names and bus numbers from the Port Authority database), and 1131 out-of-domain utterances from the Arctic database Kominek and Black [2003], all recorded by the same male, native speaker of North-American English. In order to increase the quality of the synthesized utterances, most of the generated prompts are complete sentences, whereas human participants in a conversation typically make use of sentence fragments (e.g. for confirmation). This results in very high quality synthesis (very often mistakable for a recording of a human speaker) but sometimes long and cumbersome system prompts. Some of the

speech generation templates and all the work to build the limited domain synthesis voice were done by Brian Langner, and these components were kept unchanged throughout this thesis work.

3.4 Public Deployment and Performance

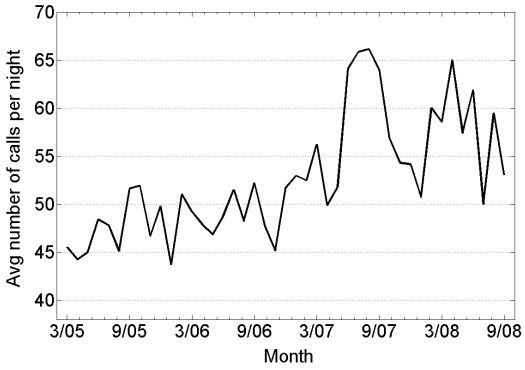
3.4.1 Call traffic

The average number of calls reaching Let's Go is about 40 on weeknights and 60 on weekend nights. Average daily call traffic oscillates between 30 and 60 (see Figure 3.5(a)), depending on seasonal variations and special events such as schedule changes. We also found that the average daily traffic for March 2006 was about 10% higher than for March 2005. One possible explanation is that some callers for whom the system worked well got used to calling outside of normal business hours (i.e. when the system is active), which was not the case early on since, before the system went public, out-of-business-hours callers would only get a recorded message asking them to call back the next day. The average length of a dialog is 14 turns. However the distribution of dialog turn lengths, shown in Figure 2, is bi-modal, with a first peak at 0 turns (10% of the dialogs) and a second one around 10 turns. This reflects two types of user behavior, with part of the population hanging up as soon as they realize they are not speaking with a human, while the other part at least attempts to get some information. Given all the necessary confirmations, the minimum number of turns necessary to get schedule information is six.

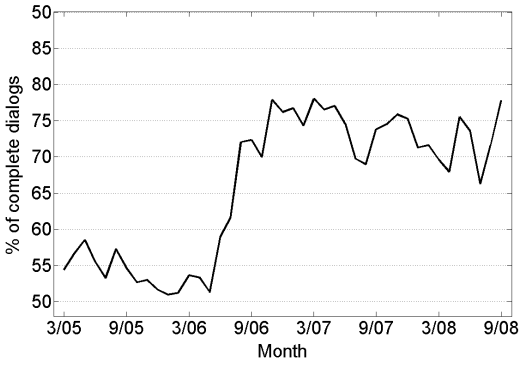
3.4.2 Dialog completion rate

In order to have a running estimate of success rate without having to manually label each call, we used the following heuristic: a dialog is marked as complete if the system obtained enough information from the user to either provide a result (by performing a database look-up) or notify the user that their request is out of coverage (e.g. it is for a bus route that we do not deal with yet). We call the ratio of complete calls to the total number of calls "dialog completion rate" (DCR).

To understand how DCR correlates with actual task success, two annotators listened to a subset of 387 dialogs and labeled each of them as a success if the system fulfilled the caller's need or a failure otherwise. Even to a human observer, this annotation is not always obvious. For example, the caller might appear to change their request during a dialog. To improve annotation coherence, the annotators first independently annotated 147 dialogs.



(a) Average daily call volume.



(b) Average Dialog Completion Rate.

Figure 3.5: Evolution of call volume and system performance between March 2005 and September 2008. The acoustic and language models of the speech recognizer were re-trained in the summer of 2006.

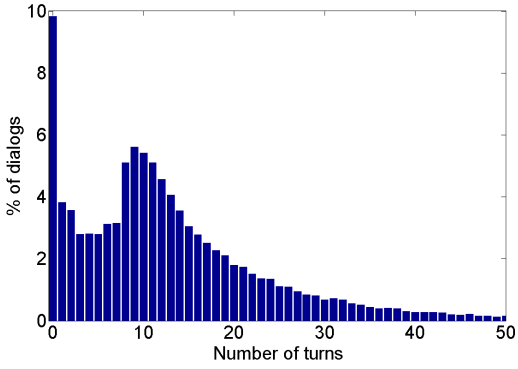


Figure 3.6: Distribution of the number of user turns per dialog.

They then discussed and resolved the discrepancies. Finally, they independently annotated the remaining 240 dialogs. On this second subset, Cohen's Kappa coefficient was 0.88.

Overall, complete dialogs have a 79% success rate. In the remaining 21% of the dialogs, the system provided the user with schedule information (hence the dialog was complete), but that information did not correspond to the caller's request. Note that by construct, incomplete dialogs are necessarily failures (since no information was given to the user). Based on this, an estimate of task success rate could be derived from DCR.

Hereafter, we only report task success rate when dialogs were manually labeled. In all other cases, we report DCR. The evolution of DCR over the past year is shown in Figure 3.5(b). The significant increase in DCR observed in summer 2006 corresponds to speech recognition models retraining (see section 3.3.1).

3.5 Turn-Taking in the Let's Go System

3.5.1 Baseline Endpointer

Let's Go, as with most current dialog systems, relies on an energy-based voice-activity detector (VAD) to identify pauses in the user speech and considers a user turn finished when a pause lasts more than a fixed threshold (700 ms). Although barge-in from the user is allowed at certain points in the dialog, the system does not have rich turn-taking management abilities and therefore imposes a fairly rigid model of turn-taking to the dialog. For example, the user backchanneling to the system as a response to confirmation requests is often misinterpreted as a barge-in and leads to the system interrupting itself when it should not. Also, if users introduce long pauses within their utterances, they are likely to be misinterpreted as turn boundaries, leading to confusion and misrecognitions.

3.5.2 Human-Computer Dialog Corpora

In order to quantify the amount of turn-taking issues, their impact on the dialog, and to compare turn-taking behavior in human-computer and human-human dialogs, we collected and annotated two sets of dialogs, both in early April 2005. The first set contains all the calls received on April 1st, 2nd and 3rd. There are 142 calls in total, of which 40 were excluded because they were too short (less than 5 user turns), or did not contain any speech directed to the system. This first set of 102 calls, hereafter referred to as the HC1 corpus, is a representative sample of calls to the system at that point in time. The hand-

	HC1	HC2	HH
Nb of dialogs	102	57	103
Average dialog duration (in min)	4.3	2.5	1.2
Avg nb of caller turns/dialog	19.8	12.5	11.0
Word error rate	60%	28%	-
Task success rate	54%	93%	-

Table 3.3: Overview of the human-computer and human-human corpora.

labeled success rate for HC1 is 54%. A second set was assembled by collecting complete dialogs with fewer than 15 user turns. These calls represent 12% of all the calls received between April 1st and 10th. This set, hereafter called HC2, has a success rate of 93% and is therefore representative of "good" dialogs, where speech recognition performed reasonably well and no other major problem occurred.

Table 3.3 shows an overview of the two corpora.

3.5.3 Turn-Taking Failures

For each failure, we counted the proportion of dialogs in which it occurred. The most frequent failure is when the system misinterprets a noise or a backchannel from the user as a barge-in and wrongly interrupts its current turn. While we disabled barge-in on crucial turns (e.g. when giving the results of the query), we still allow the user to barge in at many points in the dialog. While this allows a swifter interaction for expert users, it has a significant cost as this failure appeared in more than half of the dialogs (52%). Next in frequency (47%) is the system failing to take a turn, usually due to inaccurate endpointing (e.g. the system does not endpoint because of background noise). Third is the converse of the first one, namely the system failing to interrupt itself when the user actually attempts to barge in. This generally happens on prompts where barge-in is intentionally turned off, and shows that this option is not optimal either since users can get frustrated if the system does not respond in a timely fashion. Finally the last two failures occur when the system starts speaking when it should not, right after its own turn (System takes extra turn), usually due to a noise misinterpreted as a user turn, or in the middle of a user turn, usually by misinterpreting a hesitation pause as a turn boundary.

Overall, turn-taking failures occurred more frequently than we had anticipated, with 85% of the calls containing at least one such failure and, on average 3.8 failures per call.

Failure type	Frequency of occurrence (% calls)
System wrongly interrupts its turn	52.0%
System fails to take a turn	47.1%
System fails to yield a turn on user barge-in	43.1%
System takes extra turn	39.2%
System wrongly barges in on user	15.7%

Table 3.4: Frequency of occurrence of five turn-taking failures.

	Operator			Caller		
	Avg (s)	StdDev (s)	Nb Cases	Avg (s)	StdDev (s)	Nb Cases
Vocalizations	1.9	2.7	1423	1.5	1.5	1677
Simult. Speech	0.4	0.2	78	0.4	0.2	113
Pauses	1.6	2.9	330	0.9	1.5	641
Sw. Pauses	0.7	3.0	759	1.0	2.0	570

Table 3.5: Average State Duration and Standard Deviation in the HH Corpus

In addition, inspection of the dialogs showed that 10% of them broke down mainly for turn-taking reasons, which represents about 20% of the failed dialogs.

3.5.4 Comparison of Human-Human and Human-Computer Dialog Rhythm

Data Analysis

In addition to analyzing obvious turn-taking failures, we also wanted to compare the general turn-taking behavior of the system to that of a human operator in similar dialogs. To that end, we annotated a corpus (HH) of 103 recordings of telephone calls to Customer Service human operators at the Port Authority of Allegheny County. We selected them from a database of 3000 calls provided by the Port Authority, among calls that 1) dealt with scheduling information, 2) were less than 6 minutes long. These criteria were chosen to get dialogs that match, as closely as possible, dialogs with the system. In particular, we excluded calls unrelated to bus schedules, as well as very long calls which usually contain social interaction, small talk, and other out-of-task speech.

	System			Caller		
	Avg (s)	StdDev (s)	Nb Cases	Avg (s)	StdDev (s)	Nb Cases
Vocalizations	1.4	1.6	2257	1.5	1.7	658
Simult. Speech	1.4	1.4	89	1.6	1.4	10
Pauses	1.6	2.1	1610	1.3	1.5	63
Sw. Pauses	1.5	1.8	549	1.6	3.2	582

Table 3.6: Average State Duration and Standard Deviation in the HC2 Corpus

To evaluate the rhythmic differences between human-computer and human-human dialogs, we segmented and labeled the HC2 and HH corpora according to dyadic conversation states [Jaffe and Feldstein, 1970]. At each point in time, the dialog is in one of 8 dyadic states, 4 for each speaker: vocalization (VOC: "a continuous sound by the speaker who has the floor[2] "), pause (PAU: "a period of joint silence bounded by vocalizations of the speaker who has the floor"), switching pause (SWP: "a period of joint silence bounded by vocalizations of different speakers"), and simultaneous speech (SSP: "a sound by a speaker who does not have the floor during a vocalization by the speaker who does"). In the following, we call "a state" or "a dyadic state" a stretch of time during which the label remains the same.

We labeled in two passes. First, we manually marked the regions where each participant was speaking. This gave us the floor possession information, along with the periods of simultaneous speech, which could not be identified automatically from a single-channel recording. Then, we semi-automatically detected the pauses within each speaker's utterances. This second pass was performed using a Matlab script which looked for energy peaks within a 250 ms window and marked the window as speech or pause based on an energy threshold. Because the recording conditions varied widely between dialogs, we manually adjusted the energy threshold for each dialog so as to match as closely as possible our perception of pauses in the conversation. The final result of this phase is, for each dialog, a sequence of dyadic states. Note that our state definitions are such that simultaneous speech is attributed to the speaker who has the floor at the time the overlap occurs, and that a switching pause is attributed to the speaker who had the floor before the pause.

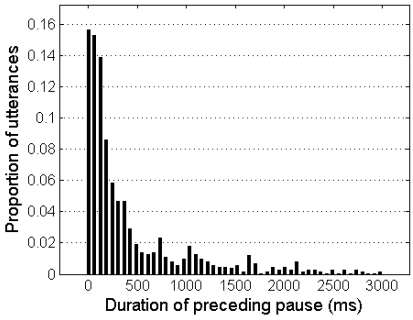
[2] Following Jaffe and Feldstein, we define "floor" in the following way: "The speaker who utters the first unilateral sound both initiates the conversation and gains possession of the floor. Having gained possession, a speaker maintains it until the first unilateral sound by another speaker, at which time the latter gains possession of the floor."

Frequency of Occurrence of Dyadic States

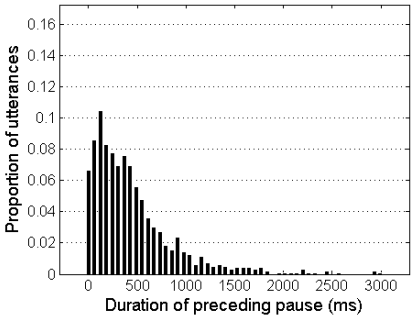
As can be seen in Tables 3.5 and 3.6 the number of occurrences of the 8 dyadic states is not only different in absolute value (which can simply be explained by the different amounts of data in each corpus), but also in its distribution across states. Hence, while the number of occurrences of simultaneous speech and switching pauses for the operator/system are similar across corpora, the number of pauses, and to a lesser extent of vocalizations, is larger in system utterances. This reflects the fact that, by design, the system is more verbose than a human operator, often uttering several sentences at a time, and pausing between them. This is, for example, the case in the introduction and when giving results (see Figure 3.1). Another difference is that callers produce many fewer vocalizations and pauses when speaking to the system than to a human. This is the result of the constrained nature of the dialog with the system, where users are only asked short answer and yes/no questions. One can assume that a mixed-initiative system would yield longer user responses with more vocalizations and pauses.

Average State Durations

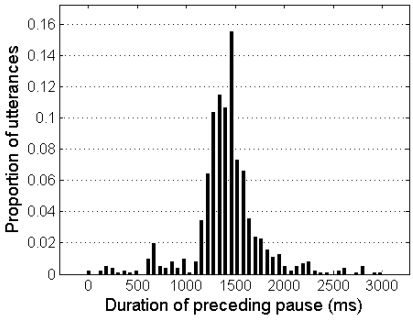
The differences in average duration between the human-human and human-computer corpora are statistically significant ($p < 0.001$ using a two-sample with unequal variance, two-tailed t-test) for all states except operator/system pauses and user vocalizations. In particular, periods of simultaneous speech are longer in the human-computer corpus, both for the system and the user. This reflects the fact that in task-oriented human-human conversations, overlapping speech is to a large extent limited to backchannels and very short overlaps at turn transitions. On the other hand, we have already observed that in many cases, the system fails (or does not attempt) to detect barge-in from the user, resulting in longer, unnatural, simultaneous speech. Second, pauses in user utterances are longer when speaking to the system rather than to a human. One explanation is that among the relatively rare cases where users do pause in their utterance, a significant number are due to the system failing to take a turn (IGN_YLD). The user then repeats their utterance or tries to reestablish the channel (e.g. by saying ”hello?”) after an unnaturally long pause. Finally, switching pauses are longer in the human-computer situation, both for the system (compared to the operator) and the user. Figure 3.7 shows the distribution of the duration of switching pauses for both participants to human-human and human-computer dialogs. First, it is clear that the system takes more time to respond than any other participant. Second, when talking to the system, human callers also take more time when they are dealing with a system than when it is a human operator. In addition, a closer look at this distribution of switching pauses preceding caller turns in HC2 shows that this ditribution appears



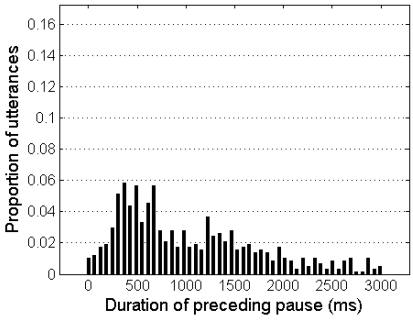
(a) Operator in HH



(b) Caller in HH

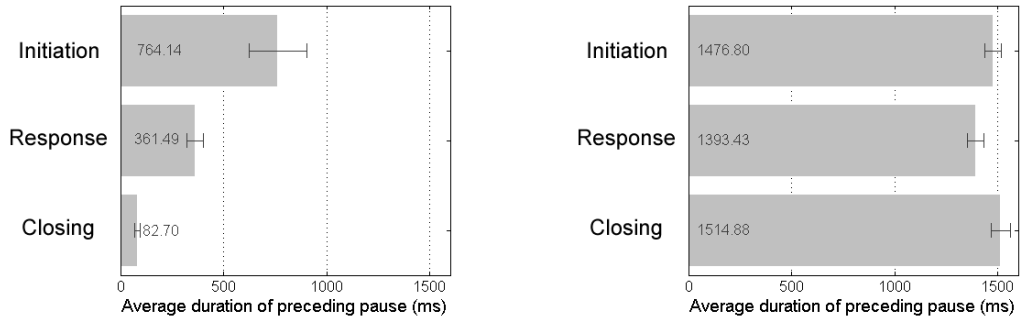


(c) System in HC2



(d) Caller in HC2

Figure 3.7: Histograms of the duration of the switching pauses preceding utterances by one of the participants in the HH and HC2 corpora



(a) Operator in HH

(b) System in HC2

Figure 3.8: Average duration (with error bars) of pauses preceding different types of dialog moves

to be bimodal. By looking at the data, we can explain this by the two most frequent types of answers that callers have to provide to the system: for answers to wh-questions (e.g. ”Where do you want to go to?”), the average preceding pause is 1341 ms, whereas for answers to confirmation questions (e.g. ”You want to go to the airport. Did I get that right?”), the average duration is 574 ms.

Interaction between Discourse Structure and Turn-Taking

To better understand the nature of the differences in switching pauses between the HH and HC2 corpora, we labeled system/operator utterances with dialog moves, which were grouped into four main discourse functions: opening, initiation, response, and closing. Initiation and response moves follow the classification proposed by Carletta et al. [1997] and were already used to study the timing of turn-taking in Matthew Bull’s thesis [Bull, 1997]. An initiation move is an utterance that introduces new expectations to the dialog (e.g. asking a question, setting a goal). A response move is one that attempts to fulfill such an expectation (e.g. acknowledgment, response to a yes-no or wh- question). Additionally, we distinguish opening and closing moves (which correspond to utterances such as ”Hello” and ”Thank you”, ”You’re welcome”, and ”Goodbye”) from other moves because they typically do not imply any transfer of information and correspond more to the ”pro-

tocol" people use when talking over the phone. In particular, because we are interested in switching pauses occurring before different types of moves, we will not address opening moves here. When an utterance contained several successive moves (e.g. an acknowledgment followed by a yes-no question), we used the move that appeared first since we assume that the pauses immediately preceding a turn are mostly affected by what comes first in the turn.

Figure 3.8 shows the average duration of pauses preceding different types of move for the operator in HH and the system in HC2. Besides the previously noted fact that the operator is always faster than the system, these graphs show that the response time of the operator depends greatly on the type of dialog move, which is not the case for the system. The result on HH is similar to what Bull and Aylett [1998] found on the HCRC Map Task corpus, namely that inter-speaker intervals are longer between conversational games (i.e. before initiation moves) than within games (i.e. before response moves). In addition, we find that closings require the shortest response time by far (almost a tenth of initiation moves). This is probably due to their predetermined nature (thus not requiring much planning from the speaker). On the other hand, system response time in HC2 is by and large dictated by processing that is independent of the structure of the dialog (e.g. endpointing, dialog state computation, synthesis...). This results in almost constant response times.

3.5.5 Discussion

Our experimental results suggest that the current approach to turn-taking in spoken dialog systems can lead to suboptimal interaction. First, many turn-taking failures occur in human-computer dialogs, and most of them would only appear in human-human conversation in the presence of a severely degraded channel (e.g. so noisy that one cannot always tell whether the other participant has finished speaking or not). Second, endpointing mechanisms based on pause detection alone result in significantly longer response times than human performance. Third, whereas human-human conversation presents a rhythmic pattern related to its meaning and discourse structure, our system (and, to the best of our knowledge, all dialog systems today) have mostly constant reaction time. The extent to which this fixed, unnatural rhythm hurts the dialog (e.g. by annoying the user) remains an open question. In any case, this rigidity reflects the fact that turn-taking mechanisms in dialog systems do not use information from the higher levels of conversation (structure and meaning).

The turn-taking behavior of human callers is also significantly affected by that of the system. There are two explanations for this. One is that humans adapt to the rhythm imposed by the system consciously or unconsciously, as has been observed by Darves and

Oviatt [2002]. This, in itself, is not a problem as long as user adaptation to unnatural patterns does not hurt the quality of the interaction. A second reason for the difference in human response time could be that users need more time to process system utterances than human operator utterances. This could be due to the fact that system prompts are worded in an unexpected way. Also, some system questions, even if they are naturally worded, could come at unexpected times for the user, for example following misrecognitions (as in the following example dialog: ”User: I want to go to the airport. System: Going to downtown. Did I get that right? User: ...”). The quality of speech synthesis, in particular its prosody, could also make comprehension and endpointing more difficult for users.

Chapter 4

Olympus 2: a Multi-Layer Spoken Dialog System Architecture

4.1 Summary

In this chapter, we describe a new, multi-layer architecture to build task-oriented spoken dialog systems. Implemented as a new version of the Olympus architecture, it features a new version of the RavenClaw dialog management framework, which explicitly takes into account the conversational floor, as well a new component, the Interaction Manager, which handles low-level reactive behavior and acts as an interface between the real world and the abstract representation used in the dialog Manager. The feasibility and practicality of the approach was confirmed by porting the Let's Go bus information system, a deployed information access system, to the new architecture.

4.2 Levels of Dialog Processing

Spoken dialog unfolds at many different levels simultaneously. When we talk, we are at the same time producing sounds, uttering words, performing dialog acts such as making a statement or asking a question, and exchanging ideas, among other levels of action. Typically in a dyadic conversation, the participant who has the floor produces content while the other perceives it (although both can do both at the same time, as is the case with backchannel feedback). Consider for example the following extract from a dialog between a human user and a hypothetical spoken dialog system:

User: I want to go to Miami.
System: Going to Miami, right?
User: Yes.

As the user speaks the first utterance, the system perceives it as streams of sounds, phonemes, words, and finally meaning. At this level, a user utterance is anchored in time: it has a start, an end, and a duration. Yet at another level, an utterance, or more precisely a semantic unit, is a single contribution from the user to the dialog. The utterance "I want to go to Miami" as a whole and its associated meaning, are qualitatively different from the partial utterances that emerged as the user spoke them ("I", "I want", "I want to", ...).

Conversely, the system utterance "Going to Miami, right?", is at the same time an atomic action (asking the user to confirm that their destination is Miami), and a composite result of producing words, phonemes, and sounds with a beginning, an end, and a duration. We thus see the emergence of two realms of dialog events and actions, one, which we call the concrete level, that is concerned with how semantic units are constructed, and the other, the symbolic level, where these semantic units are integrated in the general dialog context. While further distinction is possible in both realms, for example between phonemes, words, and semantic units at the low level, and between semantic units, turns, and discourse segments at the high level, the separation that we draw is fundamental in that concrete events and actions are typically continuous, in terms of their timing as well as of other properties such as pitch, while symbolic events and actions are discrete.

A conversational agent, whether human or artificial, can make decisions and trigger actions at both levels. However, these decisions differ in nature. Decisions at the concrete level are reactive. They involve a short, potentially subconscious, perception-action loop and trigger reactive behavior such as stopping to speak when being interrupted or nodding to indicate that one is listening. In contrast, behavior controlled at the symbolic level is deliberative. It comprises typical discourse phenomena such as grounding and reference resolution. This is where the agent keeps track of dialog context and plans the next utterance. The two levels operate to a large extent asynchronously: the symbolic level can integrate events and plan future actions just as the concrete level is perceiving and executing the current ones. Yet, in order to explain such simple and pervasive phenomena as the fact that the agent waits for the answer after asking a question, there needs to be some locking mechanism that prevents the symbolic level from triggering actions at inappropriate times (e.g. when the other participant speaking is claiming the floor). We will see that the conversational floor can be thought of as such a synchronization mechanism.

We designed a generic software architecture for spoken dialog systems that captures the concrete / symbolic dichotomy, as well as the role of the conversational floor at both

levels. The key aspects of the architecture are given in the next section. Section 4.4 presents the application of the new architecture, implemented as Olympus 2, to the Let's Go system, and Section 4.5 compares our approach with the other multi-layer architectures introduced in chapter 2.

4.3 Architecture Overview

4.3.1 Two Layers of Representation

Conceptually, our architecture distinguishes two layers of processing. At each layer, we define events, i.e. observations about the real world, and actions, i.e. requests to act upon the real world. The lower layer, the closest to the real world, corresponds to the concrete level of processing introduced in section 4.2. This layer consists of real-time events and actions with continuous properties. An example of a concrete event is the transition from silence to speech as detected by the voice activity detector. A partial recognition hypothesis from the speech recognizer and the associated parse also constitute a concrete event. The top layer is the domain of symbolic events and actions with typically discrete properties. A full utterance with its parse make up a symbolic event. The core components of the architecture perform three types of tasks:

Event composition They accept events at one level and produce events at a higher level.

Action decomposition They accept actions at one level and produce actions at a lower level.

Control They produce actions at a level in response to events at the same level.

The interface between the real world and the concrete layer is a set of sensors and actuators. No control happens at this level. The interface between the lower and top layers is a module called the Interaction Manager (IM). In addition to event composition and action decomposition, the IM controls reactive behavior that does not involve high-level cognition (e.g. stopping speaking when the user interrupts). Finally, within the top layer, the Dialog Manager (DM) plans symbolic actions based on symbolic events. Being at the highest level of the architecture, the DM does not perform any composition/decomposition.

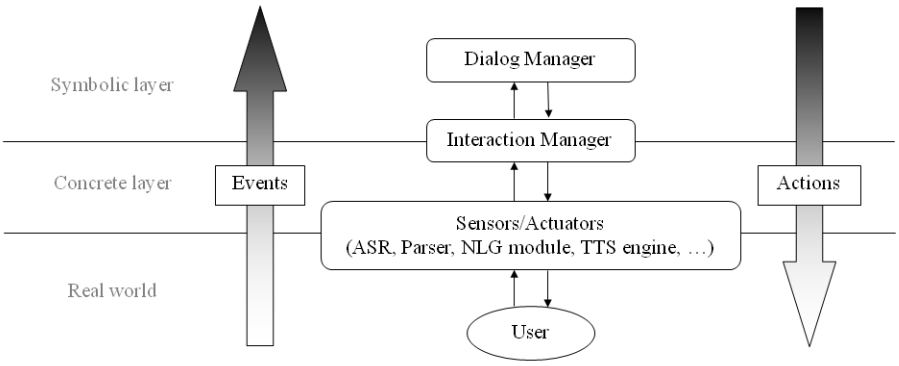


Figure 4.1: Overview of the proposed architecture.

4.3.2 Sensors and Actuators

Sensors and actuators are the point of contact between the real world and the dialog system. Their role is to translate physical phenomena into concrete events, and concrete actions into physical phenomena. Typical examples of sensors in our systems are the voice activity detector and the speech recognizer. In multimodal systems, other sensors such as a touch screen or a gaze tracker can be used. One characteristic of these sensors is that they stream real time information to the system. For example, the speech recognizer does not only produce the final recognition result for each user utterance but also partial hypotheses as the user is speaking. Some sensors have an internal state that controls their behavior and is updated by the Interaction Manager. For example, the speech recognizer's internal state indicates 1) whether the recognizer should be decoding audio or ignoring it (i.e. whether an utterance has been detected), and 2) which language model the recognizer should use, based on dialog state.

In speech only systems, the only actuator is the speech synthesizer, while multimodal systems can feature other actuators like a talking head, a graphical user interface, or a robotic arm. In addition to performing their physical actions, actuators need to inform the system in real time of their status. For example, the speech synthesizer must indicate the precise time at which it starts and finishes speaking a given utterance.

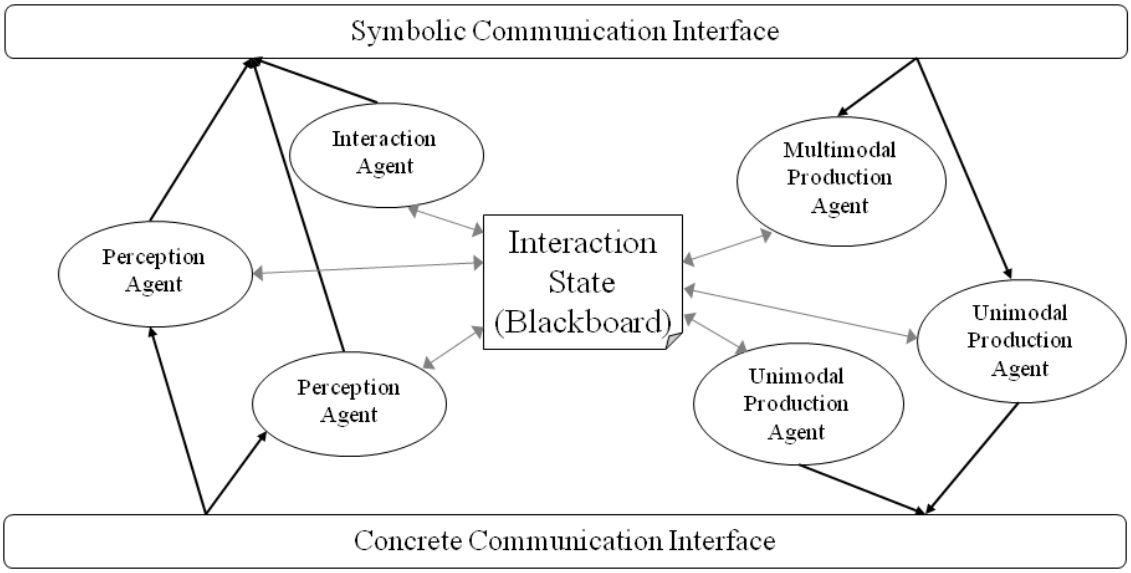


Figure 4.2: Internal Structure of the Interaction Manager.

4.3.3 Interaction Management

The Interaction Manager acts both as the interface between the concrete and symbolic levels of processing, and as the controller of the system's reactive behavior. In particular, it sends appropriate dialog state and floor update events to the Dialog Manager. In order to achieve these goals, the IM needs to:

1. react to symbolic actions and concrete events
2. integrate a variety of modalities and sensor/actuator types
3. operate in real time

Our implementation of the IM, named Apollo, fulfills these requirements (see Figure 4.2). Two interfaces handle communication of actions and events with the DM (symbolic communication interface), and the sensors and actuators (concrete communication interface). Between these two lies a set of agents, each of which handles unimodal or multi-modal perception or production. All agents can read and write to a central blackboard object called the Interaction State. Typical agents are the *Listener Agent*, which

handles events from the voice activity detector and speech recognizer, and the *Speaker Agent*, which handles actions on, and events from, the speech synthesizer. Let us consider our example from section 4.2:

> User: I want to go to Miami.
> System: Going to Miami, right?
> User: Yes.

While the user is speaking the first utterance, the IM receives events from the VAD (speech/silence transitions) and ASR (partial recognition results), and sends these events to the Listener Agent. Based on these events, the Listener Agent updates the Interaction State by, for example, marking the floor as belonging to the user. The Listener Agent uses the Interaction State to decide whether the user has released the floor (i.e. finished speaking their turn) or not. Once a complete utterance has been recognized, the Listener Agent updates the Interaction State (marking the floor as free), and sends an event to the DM containing the semantics of the new user turn. Once the floor has been marked as free, the Speaker Agent can start sending utterances to the TTS to respond to the user (e.g. fillers), or it can wait for a symbolic action request to come from the DM that contains the system response (in this case, an explicit confirmation prompt). While the system is speaking, the Listener Agent is in charge of detecting when the user attempts to interrupt it. If, for example, the user responds "Yes" before the system finished speaking the confirmation prompt, the Listener Agent updates the Interaction State accordingly, which then leads the Speaker Agent to stop the playback of the system prompt.

In a multimodal scenario, one could add an agent in charge of, for example, detecting user gestures (a Gesture Perceiver Agent), and updating the Interaction State accordingly. Multimodal fusion would then be performed by an integration agent that would use information from both speech and gesture contained in the Interaction State to send a symbolic event to the DM capturing the semantics of the user action. For example, if the user said "Give me this cup" while pointing at a specific cup on a table, the integration agent would generate an event that captures the fact that the user wants the system to give them the specific cup they point at.

In addition to the events it receives from the sensors, the concrete communication interface is also in charge of generating a pulse event, which reaches all agents and allows to react not only to specific events when they occur but also to delays between events (e.g. to wait for a given amount of time after the user finished speaking before taking the turn).

Agents that handle different modalities, such as the Listener and Gesture Perceiver Agents above, can be developed independently and later combined with an integration agent. While the use of the blackboard guarantees that any agent has access to information

from all the other agents, it allows agents to use state information when it is available but still function when it is not (e.g. information from a gaze tracker could be optionally used in an embodied agent).

Overall, the architecture fulfills the above-mentioned requirements through 1) the symbolic and concrete communication interfaces, 2) its multi-agent, distributed nature, and 3) both its simplicity (which allows efficiency) and the use of pulse events to allow reaction at any time, based on the current interaction state.

4.3.4 Dialog Management

Overview

Dialog management is performed by an extended version of the RavenClaw dialog management framework [Bohus and Rudnicky, 2003]. To provide the necessary background to our own work, we provide in this section a brief overview of the core functionalities of RavenClaw, as designed and implemented chiefly by Dan Bohus. The reader is invited to refer to previous work by Bohus and Rudnicky [2003] and Bohus [2007] for more details. A RavenClaw dialog manager is composed of two elements: a task-independent engine and a task specification. The latter is a hierarchical plan that decomposes the main goal of the dialog into subgoals and so on, recursively. One example of such a task tree is given in Chapter 3 (see Figure 3.4). Nodes of the tree are called agents. Each internal agent (also called agency) represents a (sub)goal that can be decomposed further, while terminal agents represent atomic dialog acts. There are four types of terminal agents:

Inform Agents provide information to the user.

Request Agents request information from the user and capture the expected response.

Expect Agents capture expected user inputs, without performing any dialog act.

Execute Agents perform non-conversational actions such as database look-up.

During execution, RavenClaw plans agents by pushing them on a stack and executing the agent at the top of the stack.

The default behavior is as follows. When an agency is executed, it pushes its first non-completed child onto the stack. When an agent completes its goal, either by performing its dialog act for leaf agents, or by having all of its children's goals completed for agencies, the agent gets popped off the stack. Thus this default behavior amounts to a depth-first

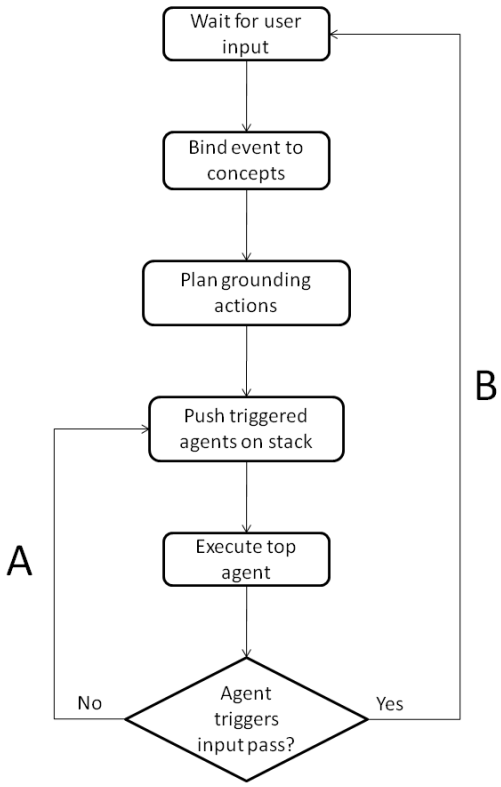


Figure 4.3: Main execution loop of the original RavenClaw dialog manager as proposed by Bohus and Rudnicky [2003].

left-to-right traversal of the task tree. This behavior can be altered by adding preconditions and completion criteria to agents, as well as having some agents being triggered by some conditional expression (usually corresponding to the user uttering a specific command, like ”HELP”). When an agent’s triggering condition becomes true, it is pushed onto the stack, thus introducing a new subdialog. Once that subdialog is complete (i.e. the triggered agent has achieved its goal), the triggered agent is popped off the stack and the dialog resumes in the context it was in before the trigger. These mechanisms, and others, allow RavenClaw to handle a wide range of interaction styles, from system initiative, to command and control, to mixed initiative.

Figure 4.3 is the flowchart for the main execution loop of the DM. When a user input reaches the DM, RavenClaw matches it against the concepts expected by the agents

currently in the stack, starting from the top, performing what we call concept binding. Next, grounding actions are planned by the DM and, if necessary, corresponding agents are pushed on the stack. Dialog agents whose trigger conditions are true are also pushed onto the stack. The agent sitting on top of the stack is then executed. As explained above, this generally leads to more agents pushed onto the stack and/or prompts being sent to be spoken by the TTS. If that agent is marked as requiring an input pass, the system waits for the next user input and restarts from the beginning (loop B). If that agent does not require an input pass (e.g. an agency, inform agent or database look-up agent), then loop A is executed again. At any time, agents whose completion criterion is true are popped from the stack.

The Conversational Floor

We extended RavenClaw to accommodate general events (as opposed to only user inputs) and better account for turn-taking issues, in particular for the fact that the system waits for a user response after asking a question.

We introduced a new ternary variable that tracks the conversational floor. Its possible values are USER, SYSTEM, and FREE[1].

Floor transitions happen in two ways. First, incoming events can be marked as modifying the floor. For example, an event that corresponds to the end of a user utterance, will set the floor to FREE. However, system questions (produced by Request agents) are typically marked to set the floor to USER once they have been asked. This is what makes the system wait for the user answer, or another event that would change floor status, such as a time out. Second, the floor can be directly affected by the DM's own actions. When RavenClaw executes an Inform or Request agent, the floor becomes SYSTEM, until the IM sends the notification that the system utterance has been completed or interrupted.

All agents are marked as requiring or not requiring the floor. By default, all Inform and Request agents require the floor, while Expect and Execute agents do not. While the floor is not FREE, new events are accepted and bound, and agents can be triggered and pushed on the stack, and even executed if they do not require the floor. But the floor-requiring agents are not executed. Their execution resumes when the DM receives an event that sets the floor to FREE (such as the end of a system prompt or of a user utterance).

The distinction between agents requiring and not requiring the floor allows RavenClaw to be at the same time synchronized with the user (e.g. waiting after asking a question),

[1]We do not deal with cases where both participants claim the floor at the same time in the DM, the IM being in charge of resolving these situations at the concrete level

and performing non-conversational behavior in the background. This is important because, in systems that operate in the real world, non-conversational events can come at any time, asynchronously from the dialog. For example, in a human-robot dialog, the robot visual sensor might detect the presence of an unexpected object (e.g. a vehicle heading toward the user) while the user is speaking. Even though the floor might not be FREE, this event will be processed and bound by the DM. If necessary, the detection of this object can trigger an Inform agent set to not require the floor but instead grab it, interrupting the user (e.g. to warn them of the impending collision).

Thus the conversational floor in RavenClaw offers a flexible synchronization mechanism that captures the turn-taking nature of conversation while allowing a wide range of asynchronous behavior.

Main Execution Loop

Figure 4.4 illustrates the new main execution loop of the RavenClaw engine. When a new event arrives, it is matched against the current stack and bound to appropriate concepts. Then, if the event requires grounding, i.e. if it's a user input, RavenClaw's grounding management mechanisms are used to potentially push grounding agents on the stack. RavenClaw then enters subloop C (see Figure 4.4). First, RavenClaw updates the floor based on the incoming event. Agents with trigger conditions are pushed on the stack. Finally, RavenClaw checks the agent that sits on top of the stack. If it is marked as not requiring the floor or if the floor is free, RavenClaw executes the top agent and returns to the beginning of loop C. On the other hand, if the agent is marked as requiring the floor and the floor is either USER or SYSTEM, the agent stays on top of the stack but is not executed. Instead, RavenClaw returns to the initial state of loop D, waiting for the next event to come from the IM. Note that, during this period, the whole stack below the floor-requiring agent is on hold. That is to say, the non-floor-requiring agents that are already on the stack are not executed (since the top agent is not popped off the stack). In practice, this is not a problem because the execution of these agents is typically conditioned on the top, floor-requiring, agent's completion (since they were under it in the stack). However, additional, non-floor-requiring agents can be pushed on the stack over the blocked agent, and executed.

The key difference with the original RavenClaw execution loop of Figure 4.3 is that events can be interpreted and non-conversational agents can be executed when the floor is not FREE, whereas the Input Pass strictly stops the DM execution until the next user input is received.

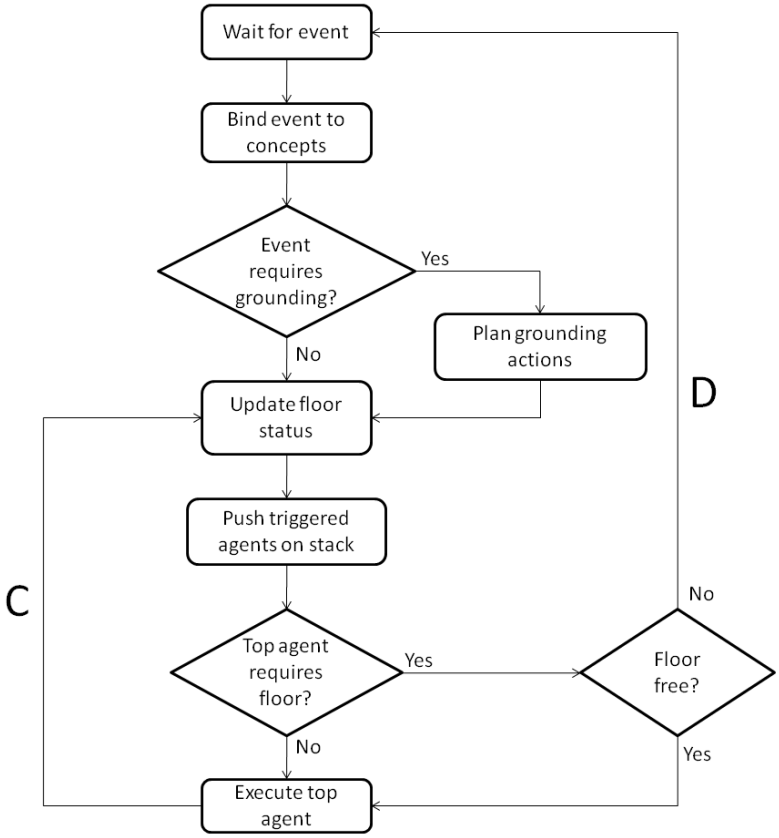


Figure 4.4: Main execution loop of the proposed approach.

4.4 Application to the Let's Go System

We implemented the proposed architecture as Olympus 2, a new version of the Olympus [Bohus et al., 2007] spoken dialog framework. By doing so we were able to reuse a number of modules and ensure the task-independence of our implementation.

We then ported the Let's Go bus information system from Olympus 1 to Olympus 2. In its first two years of operation, Let's Go handled more than 34000 calls and was progressively improved to reach a dialog completion rate of 76.7% . In April 2007, we ported the system to Olympus 2. This required only minor modifications to the domain-dependent parts of the system. Since Let's Go is a unimodal system, Apollo has only one perception agent and one production agent: respectively, the Listener Agent, which handles the ASR/NLU sensor and the Speaker Agent, which handles the NLG/TTS actuator. Initially, the turn-taking rules within the agents were hand-written so as to follow a standard behavior, similar to that of Olympus 1. Thus, for example, the system considers that the user yields the floor based on pauses of more than 800 ms. Barge-in is only allowed during certain system prompts. These rules, while simple and leaving many turn-taking issues unsolved were adopted as a baseline, and a proof of concept. Chapters 5 and 6 illustrate more advanced turn-taking models that exploit the proposed architecture. In its first three months of operation, the new version of Let's Go handled 5000 dialogs, 3738 of which have 4 user turns or more. The completion rate among these longer dialogs is 76%, almost identical to the rate in the three months preceding the switch to the new version (the difference is not statistically significant)[2]. Similarly, the average duration and number of turns per dialog have remained stable (resp. from 137.3 s to 138.8 s, and from 15.9 turns to 16.2 turns).

4.5 Discussion

Other multi-layer approaches to dialog management have been proposed. An early and important work is that of Thorisson [Thorisson, 1996, 2002]. His model is divided in three layers: the Content Layer, which deals with topics and tasks, the Process Control Layer, which deals with typical dialog phenomena (e.g. taking a turn), and the Reactive Layer, which deals with highly reactive behaviors (e.g. gazing at objects mentioned by the other speaker). Each layer has a specific target perception/production loop time (from less than 500 milliseconds for the Reactive Layer to more than 2 seconds for the Content Layer).

[2]The dip in performance that can be seen for July and August 2007 on Figure 3.5(b), down to about 70%, is the result of hardware issues that forced us to use a different telephony board with poorer audio quality

Processes on different layers communicate through two blackboards (one shared by the Content and Process Control Layers, and the other shared by the Process Control and Reactive Layers). This allows all processes to have access to any bottom-up or top-down signal, while limiting inter-layer communication to a small set of predefined messages. Unfortunately, Thorisson provides little detail on the inner workings of the Content Layer. While this is a seminal work and an influential effort towards realistic turn-taking behavior in conversational agents, it was developed largely independently of past and concurrent work on high-level dialog management. Therefore, it remains unclear how this model would work in the context of complex, task-oriented dialogs. More recently, Lemon et al. [2003] propose an architecture for task-oriented dialog systems that distinguishes a Content Layer and an Interaction Layer. The Content Layer has at its core a dialog Manager that operates on logical forms. The Interaction Layer involves lower level modules such as speech understanding and generation, as well as a Turn Manager. As in Thorisson's architecture, the two layers work asynchronously and communicate through a set of specialized data structures (e.g. a prioritized output agenda which contains the planned system utterances). This architecture captures a number of interaction phenomena, including turn taking. However, the turn-taking model reported in Lemon et al. [2003] seems to be exclusively contained in the Interaction Layer and it is not clear how the Dialog Manager handles floor issues.

The Olympus 2 architecture combines elements from both Thorisson's (the focus on turn-taking) and Lemon's (the connection to a traditional dialog management framework) work. Our dialog management model goes further than previous work by making explicit how the theoretical concept of conversational floor influences the execution of the dialog plan. This particularity constitutes a departure from the purely asynchronous models previously proposed.

To enforce turn-taking behavior, practical DMs such as the original version of RavenClaw [Bohus and Rudnicky, 2003] or COLLAGEN [Rich et al., 2002], resort to synchronization mechanisms extraneous to the planning and execution model. Bohus and Rudnicky [2003] use a separate Input Pass, while Rich et al. [2002] mention an unspecified "separate layer of discourse". These mechanisms freeze DM execution when user input is expected (e.g. after the DM executes a "question" action). During that time, the DM cannot accept nor plan further events and actions until the end of the user utterance. The floor mechanism that we propose resolves these synchronization issues in a more general and flexible way, allowing the DM to handle both synchronous events (i.e. user turns) and asynchronous events (e.g. non-conversational events and backchannels).

One of our main contributions is to provide the community with an open framework based on the multi-layer approach and to show the applicability of this approach to de-

ployed systems. The fact that Let's Go was ported to Olympus 2 with only minimal modifications to the domain-specific parts of the system (about half a day's work) confirms that systems can be built on top of reactive architectures without large overhead in terms of system design.

In the context of this thesis, the benefits of this architecture over the original Olympus architecture will become clear in the next chapters. By centralizing interaction management in one component, while separating it from dialog management, it allowed us to implement and evaluate task-independent models of turn-taking. At the same time, the "blackboard" function of the interaction manager, as a central component gathering information from on the one hand sensors and actuators and on the other hand the dialog manager, provides access to a wide range of features that can be exploited in these models. As such, the proposed architecture has fulfilled one of its main goals, namely to provide a platform for domain-independent research on turn-taking and low-level interaction.

In addition to Let's Go, the TeamTalk system [Harris et al., 2005], which handles communication within a mixed team of humans and robots, has been ported to Olympus 2. Other projects, both at CMU and at least two in other universities are currently using the architecture. We hope these efforts will shed light on the benefits that multi-layer architectures can bring to a wide range of applications, from simple information access systems to multi-participant interaction with embodied agents. In the process, theoretical as well as practical challenges will undoubtedly surface, which will extend our understanding of low- and high-level conversational phenomena.

Chapter 5

Optimizing Endpointing Thresholds

5.1 Summary

In this chapter, we propose an approach to improve the behavior of the Interaction Manager by exploiting a wide range of features made available by the architecture presented in Chapter 4. Specifically, we start by analyzing the relationship between pause distribution and a wide range of automatically extracted features from discourse, semantics, prosody, timing and speaker characteristics. These results are presented in section 5.3. Then, in section 5.4, we propose an algorithm to dynamically set endpointing threshold for each pause. Both batch and live (i.e. deployed in an actual system) evaluations are presented in section 5.5. When all features are used, the proposed method reduces latency by up to 24% compared to a fixed-threshold baseline, and performs similarly to a previous feature-based approach proposed by Ferrer et al. [2003], while generating more compact decision trees. Closer analysis showed that prosodic features do not help threshold optimization once other feature are included. The practicality of the approach and the batch evaluation results were confirmed by implementing the proposed algorithm in the Let's Go system.

5.2 Introduction

One of the central tenets of conversation, whether it be from the Conversation Analysis (CA) perspective or from the point of view of building spoken dialog systems, is the ability for a listening participant to detect when it is appropriate to respond to the current speaker. Much of Sacks et al. [1974] and subsequent CA work specifically addresses the definition

and characterization of transition relevance places (TRPs), which are those times in a participant's speech where it is appropriate (or *relevant*) for someone else to start speaking (and take the floor). While, as seen in Chapter 2, socio- and psycho-linguists have uncovered a wide variety of factors that help humans detect or anticipate the end of turns, practical spoken dialog systems have generally adopted a simplistic approach. Typically, pauses in the user speech are detected using a Voice Activity Detector (VAD) and a turn is considered finished once a pause lasts longer than a fixed threshold. This approach has the advantage of being simple, only relying on easily computable low-level features. However, it leads to suboptimal behavior in many instances. First, cut-ins happen when a turn-internal pause (TIP) lasts longer than the threshold and gets wrongly classified as a turn boundary (TB). Second, latency occurs at the end of every user turn, because the system must wait for the duration of the threshold before classifying a pause as TB. When setting the threshold, system designers must consider the trade-off between these two issues: setting a low threshold reduces latency but increases cut-in rate, while setting a high threshold reduces cut-in rate but increases latency.

To help overcome the shortcomings of the single-threshold approach, several researchers have proposed to use features from dialog. Sato et al. [2002] used decision trees to classify pauses longer than 750 ms as TB or TIP. By using features from semantics, syntax, dialog state, and prosody, they were able to improve the classification accuracy from a baseline of 76.2% to 83.9%. While this important study provides promising results showing the value of using various sources of information in a dialog system, the proposed approach (classifying long pauses) is not completely realistic (what happens when a TB is misclassified as a TIP?) and does not attempt to optimize latency. An extension to this approach was proposed by Takeuchi et al. [2004], in which a turn-taking decision is made every 100 ms during pauses. However, in this latter work the features are limited to some timing, prosody, and part-of-speech. Also the reported classification results, with F-measures around 50% or below do not seem to be sufficient for practical use.

Similarly, Ferrer et al. [2003] proposed the use of multiple decision trees, each triggered at a specific time in the pause, to decide to either endpoint or defer the decision to the next tree, unless the user resumes speaking. Using features like vowel duration or pitch for the region immediately preceding the pause, combined with a language model that predicts boundaries based on the preceding words, Ferrer et al are able to reduce latency while keeping the cut-in rate constant. On a corpus of recorded spoken dialog-like utterances (ATIS), they report latency reductions of up to 81% for some cut-in rates. While very promising, this approach has several drawbacks. First it relies on a small set of possible decision points for each pause, preventing fine optimization between them. Second, the trees are trained on increasingly smaller datasets requiring smoothing of the tree scores

to compensate for poor training of the later trees. Finally, and perhaps most importantly, these authors have investigated prosodic and lexical features, but by and large ignored other aspects of dialog, such as discourse structure, timing, and semantics.

In dialog problems other than turn-taking, dialog features have also been found useful. Most notably, a significant body of research has focused on the use of a wide range of features for confidence annotation. Confidence annotation refers to the problem of estimating, for a given user input, the probability that the system's understanding is accurate, as opposed to corrupted by speech recognition and/or natural language understanding errors. The general approach is to frame this task as a supervised classification learning problem. In a corpus of previously recorded dialogs, each user turn is labeled as correctly understood or not, and annotated with a number of features. Then some statistical classifier, such as decision trees, Support Vector Machines, or logistic regression, is trained on the corpus to predict understanding accuracy based on the features. Among the feature sources that have proved useful are: acoustics/ASR [Gabsdil and Lemon, 2004], syntax [Guillevic et al., 2002], prosody [Hirschberg et al., 2004], discourse [Higashinaka et al., 2005], and pragmatics [Gabsdil and Lemon, 2004]. Bohus and Rudnicky [2002] provide a good example of the integration of features from many different levels in a single framework. Similar features have been used to detect user's emotions in spoken dialogs [Ang et al., 2002, Liscombe et al., 2005, Ai et al., 2006].

In this chapter, we propose an algorithm that leverages automatically extracted dialog features to directly optimize end-of-turn detection thresholds. Section 5.3 describes the analysis of the relationship between pause distribution and a wide range of dialog features that are available in a standard spoken dialog system. Section 5.4 outlines a new algorithm to dynamically set the endpointing threshold for each pause. Evaluation results, both off line and in the deployed Let's Go system are given in Section 5.5.

5.3 Analysis of Endpointing in the Let's Go System

5.3.1 The Let's Go Random Threshold Corpus

The baseline Let's Go system uses a VAD based on Gaussian Mixture Models trained on previously transcribed dialogs, along with a fixed threshold on pause duration for endpointing. In order to investigate the effect of the endpointing threshold on cut-in rate and other metrics, we collected a corpus of dialogs using a wide range of thresholds. For three months spread across summer and fall 2007, the Let's Go Public system was set to randomly pick an endpointing threshold between 400 and 1200 ms at the beginning of each

dialog. This provided us with a corpus of 2839 dialogs and 39125 user turns. Of these, the first three days (from June 16 to 18) were manually transcribed and annotated for cut-ins, totaling 237 dialogs (3299 turns).

5.3.2 Automatic Cut-in Annotation

To overcome the cost of manually labeling cut-ins, we designed a heuristic to automatically mark runtime endpointing decisions as correct or cut-ins. This heuristic is based on the observation, made on the manually annotated data, that when a cut-in occurs, the user tends to resume speaking soon after the system wrongly endpointed the turn. Concretely, we mark as cut-ins endpoints for which the user resumed speaking within 1200 ms. The performance of the heuristic was further improved by excluding those turns where the user and the system were speaking at the same time to begin with, since during these turns, the user might actually interrupt their own turn to respond to concurrent system speech, leading to a short gap between two user turns that we do not consider a cut-in. The heuristic in its final form is thus:

> At a given runtime endpoint, if the user resumes speaking within 1200 ms, consider the endpoint as a cut-in, except if the user and system were speaking simultaneously.

The performance of this heuristic is given in terms of precision, recall and F-measure in Table 5.1. Because we used automatic voice activity detection, in some of these cases, the turn preceding and/or the turn following the pause are not actual user speech but rather background noise. If we exclude these noisy cases, the performance of the heuristic, given in Table 5.2, is much better. We found that the performance of the heuristic depended heavily on the preceding system prompt. Prompts come in three broad categories: Open (”What can I do for you?”), Closed (”Where are you leaving from?”), and Confirmation (”Leaving from the airport. Is this correct?”). As seen in Tables 5.1 and 5.2, the heuristic performs best on Open prompts ($F = 89.5\%$ on actual boundaries), and worst on Confirmation prompts ($F = 46.5\%$), with Closed prompts between the two ($F = 68.0\%$). However, on these last two contexts, the cut-in rate was in any case very low (resp. 3.0% and 1.8% for Closed and Confirmation prompts).

5.3.3 Thresholds and Cut-in Rates

To analyze the correlation between threshold and cut-in rate, we bin the dialogs in 8 bins according to their selected threshold: dialogs whose threshold falls between 400 and 500

Context	Number of samples	Cut-in rate		Precision	Recall	F-measure
		Real	Estimated			
Whole corpus	3299	3.1%	4.0%	58.5%	75.2%	66.4%
Open prompt	492	10.4%	9.6%	89.4%	82.4%	85.8%
Closed prompt	1335	2.7%	4.3%	46.6%	75.0%	59.1%
Confirm. prompt	1472	1.2%	2.0%	33.3%	55.6%	43.0%

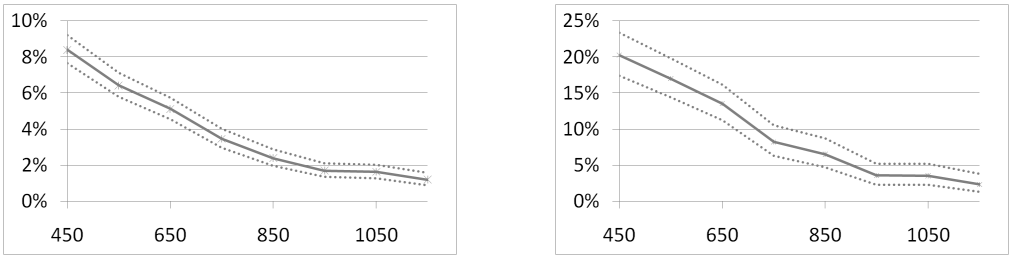
Table 5.1: Performance of the cut-in labeling heuristic.

Context	Number of samples	Cut-in rate		Precision	Recall	F-measure
		Real	Estimated			
Whole corpus	2895	3.5%	3.7%	70.6%	75.5%	73.0%
Open prompt	414	12.1%	10.1%	97.6%	82.0%	89.5%
Closed prompt	1180	3.0%	3.8%	60.0%	77.1%	68.0%
Confirm. prompt	1301	1.3%	1.7%	40.9%	52.9%	46.5%

Table 5.2: Performance of the cut-in labeling heuristic on actual speech boundaries.

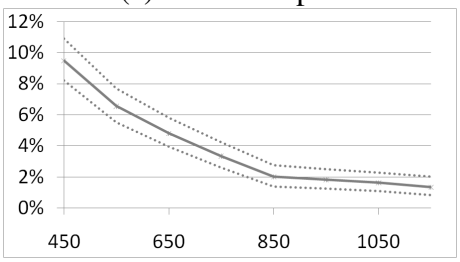
ms are grouped in the first bin (with average threshold at 450 ms), those whose threshold is between 500 and 600 ms in the second one, and so on. Figure 5.1 shows how cut-in rate varies with threshold on the whole data as well as after different types of system prompts.

As expected, we observe a strong correlation between endpointing threshold and cut-in rate in all dialog contexts. Specifically, the logarithm of the cut-in rate is linearly related to the threshold value, as illustrated in the semi-logarithmic-scale graph of Figure 5.2. The correlation coefficients R for, resp., Open, Closed, and Confirm prompts are 0.99, 0.98, and 0.99. Interestingly, cut-in rates vary across dialog contexts. Open prompts yield user responses that are more prone to cut-ins than closed prompts, and even more so confirmation prompts. This matches the intuition that responses to open prompts are more likely to be longer and contain hesitations (thus possibilities of cut-ins), whereas closed and confirmation prompts typically yield short answers (e.g. "FORBES AND MURRAY", "YES", etc). To the extreme, single word answers without any pause do not allow for any cut-in at all. These observations suggest that different thresholds (longer for open prompts and shorter for closed and confirmation prompts) should be used in these different contexts, which is the underlying principle of the approach proposed in Section 5.4.

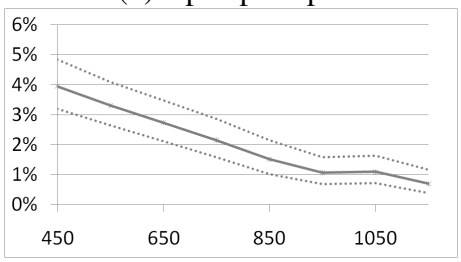


(a) Whole corpus

(b) Open prompts



(c) Closed Prompts



(d) Confirm prompts

Figure 5.1: Relationship between endpointing threshold and cut-in rate.

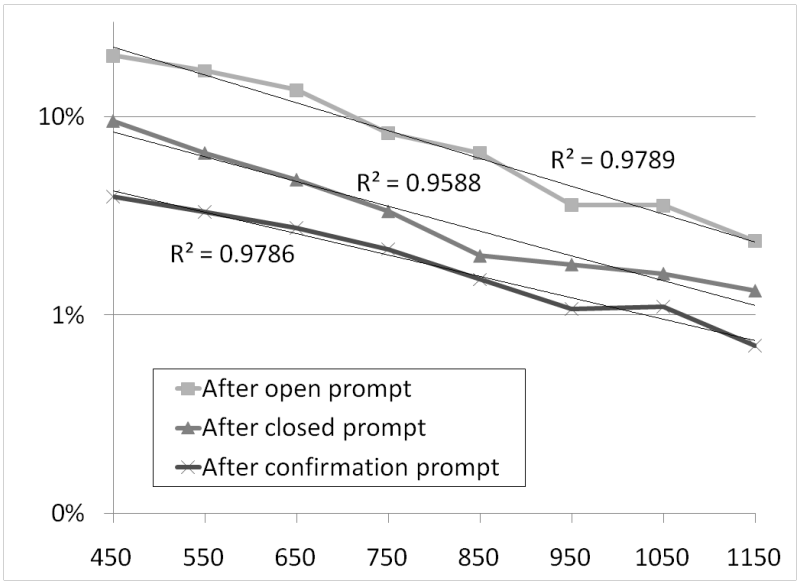
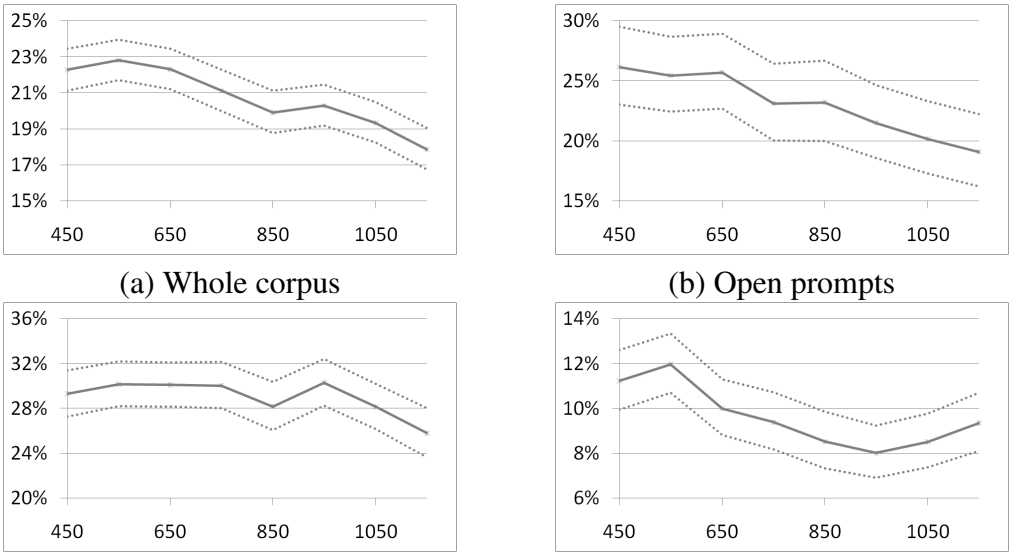


Figure 5.2: Relationship between endpointing threshold and cut-in rate (semi-logarithmic scale).



(a) Whole corpus

(b) Open prompts

(c) Closed Prompts

(d) Confirm prompts

Figure 5.3: Relationship between endpointing threshold and non-understanding rate.

Besides the confusion that they induce on the user, cut ins can have other negative effects on dialog. For example, cut-in utterances are likely to be badly recognized by the speech recognizer since they contain only fragments of sentences, whereas the recognizer's language model is typically trained on full utterances. To assess the impact of the endpointing threshold on speech recognition accuracy on the large amounts of untranscribed data in the Let's Go Random Threshold Corpus, we analyzed non-understandings. Non-understandings (also called rejects) happen when the system receives a user utterance but is not able to interpret it semantically, either because the recognition result is not parsable or because the parse does not match expected user input in the current state (e.g. "5 PM" after the system asks "Where are you leaving from?"). Figure 5.3 shows the non-understanding rate (proportion of turns that are non-understandings) as a function of the endpointing threshold. There is a general trend towards more non-understandings for shorter thresholds. One exception to this is after confirmation prompts, where non-understanding rate appears to reach a minimum of 8% for thresholds around 900 ms and then increase again for higher thresholds. This phenomenon, which we also observed in the live evaluation of our approach (see Section 5.5), can be explained by the fact that most utterances after confirmation prompts are short (mostly "YES" and "NO"). For such utterances, including a long silence after the end of user's speech (because of a long threshold)

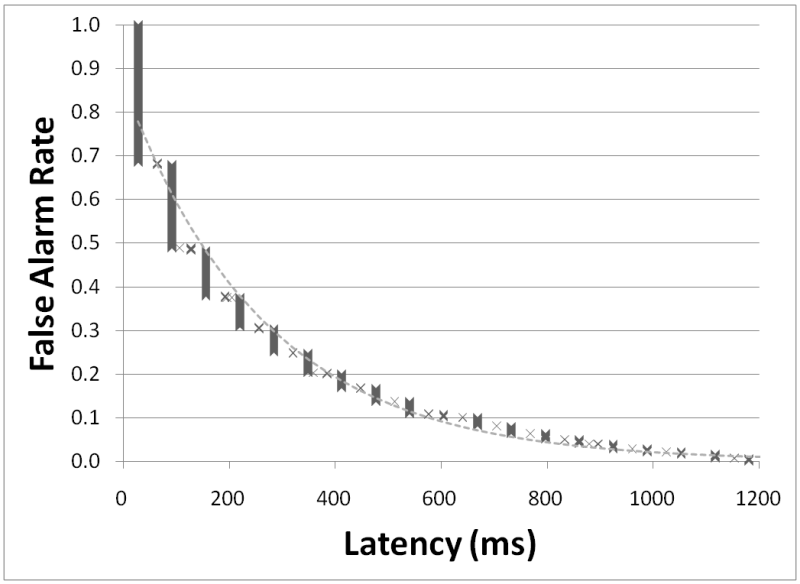


Figure 5.4: False Alarm / Latency Trade-off in the Winter Corpus.

can lead to poor recognition, particularly in the presence of background noise. Hence, not only short thresholds but also in some case long ones can hurt speech recognition.

5.3.4 Relationship Between Dialog Features and Silence Distributions

The Let's Go Winter '07 Corpus

This analysis is based on a second corpus of dialogs, collected with the Let's Go system in December 2007 and January 2008. The endpointing threshold in this corpus is fixed at 700 ms. This corpus contains 1337 dialogs and 20190 turns.

Overall there were 9471 TIPs in the corpus, which amounts to 0.55 pauses per turn. The trade-off between latency and False Alarms for the corpus is plotted in Figure 5.4. Note that False Alarms and cut-ins, although strongly related, are not identical, because False Alarm rate considers all pauses in the corpus independently, whereas in practice, once the system has detected the end of a turn, the pauses found in the remainder of the turn, if any, are irrelevant, which is taken into account when computing cut-in rate. Nevertheless, we found that the distributional properties of cut-in rates were very similar to

that of FA rates. Indeed, similarly to what we observed in Figure 5.2, this curve closely follows an exponential function (i.e. $Log(FA)$ is strongly correlated with latency, with $R = 0.99$). This stems from the fact that TIP duration approximately follows an exponential distribution, which has been observed by others in the past [Jaffe and Feldstein, 1970, Lennes and Anttila, 2002].

One consequence of the exponential-like distribution is that short pauses strongly dominate the distribution. We decided to exclude pauses shorter than 200 ms from the following analysis for two reasons: 1) they are more prone to voice activity detection errors or short non-pause silences within speech (e.g. glottal stops), and 2) in practice, 200 ms is the minimum amount of time required to detect the pause and extract the features. Once these pauses have been excluded, there are 2339 TIP in the corpus, 0.12 per turn.

Statistical Analysis

In order to get some insight into the interaction of the various aspects of dialog and pause characteristics, we investigated a number of features automatically extracted from the dialog recordings and system logs. Each feature is used to split the set of pauses into two subsets. For nominal features, all possible splits of one value vs all the others are tested, while for continuous and ordinal features, we tried a number of thresholds and report the one that yielded the strongest results. In order to avoid extreme cases that split the data into one very large and one very small set, we excluded all splits where either of the two sets had fewer than 1000 pauses. All the investigated splits are reported in Table 5.3 and 5.4. We compare the two subsets generated by each possible split in terms of two metrics:

- Boundary Ratio (BR), defined as the proportion of turn-final pauses among all pauses of a given set. We report the absolute difference in BR between the two sets, and use chi-square in a 2x2 design (TIP vs TB and one subset vs the other) to test for statistical significance at the 0.01 level, using Bonferroni correction to compensate for multiple testings.
- Mean TIP duration. The strength of the interaction is shown by the difference in mean pause duration, and we use Mann Whitney's Rank Sum test for statistical significance, again at the 0.01 level using Bonferroni correction.

We group features into five categories: discourse, semantics, prosody, timing, and speaker characteristics, described in the following sections.

Category	**Feature test**	**Data Split**	**Boundary Ratio**
Timing	Pause start time $\geq$ 3000 ms	1836 / 19260	65% / 87%
Timing	Pause number $\geq$ 2	3379 / 17717	69% / 88%
Discourse	Previous question is open	3376 / 17720	70% / 88%
Semantics	Utterance expectation level $\geq$ 1	10025 / 11071	78% / 92%
Individual	Mean pause duration $\geq$ 500 ms	1336 / 19760	72% / 86%
Semantics	Utterance contains a positive marker	4690 / 16406	96% / 82%
Prosody	Mean energy of last vowel $\geq$ 5	1528 / 19568	74% / 86%
Prosody	Slope of energy on last vowel $\geq$ 0	6922 / 14174	78% / 89%
Individual	Mean number of pauses per utt $\geq$ 3	1929 / 19267	76% / 86%
Semantic	Utterance is a non-understanding	6023/15073	79% / 88%
Discourse	Previous question is a confirmation	8893 / 12203	90% / 82%
Prosody	Duration of last vowel $\geq$ 1	1319 / 19777	78% / 86%
Prosody	Mean pitch on last voiced region $\geq$ 5	1136 / 19960	92% / 85%
Prosody	Slope of pitch on last voiced region $\geq$ 0	6617 / 14479	82% / 87%
Semantics	Utterance contains a negative marker	2667 / 18429	87% / 85%*
Discourse	Previous question is closed	8451 / 12645	86% / 85%*

Table 5.3: Effect of dialog Features on Pause Finality. * indicates that the results are not statistically significant at the 0.01 level.

Category	**Feature test**	**Data Split**	**Mean TIP Duration (ms)**
Prosody	Mean pitch on last voiced region ≥ 4	172 / 2911	608 / 482
Semantics	Utterance Expectation Level ≥ 4	2202 / 881	475 / 526
Prosody	Slope of energy on last vowel ≥ 1	382 / 2701	446 / 495
Timing	Pause number ≥ 2	1031 / 2052	459 / 504
Discourse	Previous question is open	1015 / 2068	460 / 504
Individual	Mean pause duration ≥ 500 ms	370 / 2713	455 / 494*
Prosody	Mean energy of last vowel ≥ 4.5	404 / 2679	456 / 494*
Semantics	Utterance contains a positive marker	211 / 2872	522 / 487*
Discourse	Previous question is closed	1178 / 1905	510 / 477*
Timing	Pause start time ≥ 3000 ms	650 / 2433	465 / 496*
Semantic	Utterance is a non-understanding	1247 / 1836	472 / 502*
Prosody	Duration of last vowel ≥ 0.4	1194 / 1889	507 / 478*
Individual	Mean number of pauses per utterance ≥ 2	461 / 2622	474 / 492*
Semantics	Utterance contains a negative marker	344 / 2739	504 / 488*
Prosody	Slope of pitch on last voiced segment ≥ 0	1158 / 1925	482 / 494*
Discourse	Previous question is a confirmation	867 / 2216	496 / 487*

Table 5.4: Effect of dialog Features on Turn-Internal Pause Duration. * indicates that the results are not statistically significant at the 0.01 level.

Discourse Structure

Discourse structure has been found be informative for many tasks, from confidence annotation [Bohus and Rudnicky, 2002, Higashinaka et al., 2005, Rotaru and Litman, 2006a], to emotion detection [Liscombe et al., 2005, Forbes-Riley et al., 2007], to system performance evaluation [Rotaru and Litman, 2006b]. While there are many ways to characterize the structure of dialog, researchers in spoken dialog systems, often rely on a dialog act (DA) classification scheme that characterizes how utterances by participants contribute to the advancement of the dialog. Example of generic DAs are "statement", "wh-question", "acknowledgment", etc. In accordance with previous results by Bull and Aylett [1998], we have shown in section 3.5.4 that acts with different discourse functions such as "initiation" or "response" result in different turn-taking behavior in human-human conversations.

Here, we use the dialog acts of system prompts, which are robustly available to the system at runtime, as discourse features. In the Let's Go dialogs, 98% of system dialog acts directly preceding user turns are questions[1]. Of these, 13% are open questions ("What can I do for you?"), 39% are closed questions ("Where are you leaving from?") and 46% are confirmation requests ("Leaving from the airport. Is this correct?")[2]. In terms of BR, open questions are correlated with low finality (cf Table 5.3). One explanation is that they tend to be longer (2046 ms on average, to be contrasted with 1268 ms for turns following closed questions and 819 ms for responses to confirmation questions). Conversely, confirmation questions lead to responses with significantly fewer TIPs. 78% of such turns contained only one word, single YES and NO answers accounting for 81% of these one-word responses, which obviously do not lend themselves to internal pauses. Discourse context also has an effect on TIP durations, albeit a weak one, with open questions leading to turns with shorter pauses. One possible explanation for this is that pauses after closed and confirmation questions tend to reflect more hesitations and/or confusion on the user's side, whereas open questions also yield pauses in the normal flow of speech (because responses are longer).

Semantics

In this context, Semantics refers to the meaning of the utterance being spoken, in the current discourse context. As we have seen in section 2.1.2, Oreström [1983] and Furo [2001] have found that semantics are a good predictor of turn transitions in human-human

[1]The remaining 2% belong to special cases such as certain user barge-ins.

[2]The high number of confirmations comes from the fact that Let's Go is designed to ask the user to explicitly confirm every concept.

conversations. In his work on utterance-level confidence annotation for dialog systems, Bohus and Rudnicky [2002] used as feature whether the concepts appearing in the parse of a given recognition result correspond to concepts expected by the dialog manager in the current dialog state. Bohus found this feature to be one of the most predictive of understanding errors.

We follow, Bohus and Rudnicky [2002] in our definition of semantic features. However, while Bohus was classifying full turns as correctly or incorrectly understood, we evaluate how *partial speech recognition hypotheses* match the current dialog context. Specifically, we use the latest recognition hypothesis available at the time when the pause starts, parse it using the system's standard parser and grammar, and match the parse against the "expectation agenda" that RavenClaw [Bohus and Rudnicky, 2003] maintains. The expectation level of a partial utterance indicates how well it fits in the current dialog context. A level of 0 means that the utterance can be interpreted as a direct answer to the last system prompt (e.g. a "PLACE" concept as an answer to "Where are you leaving from?", a "YES" or a "NO" after a confirmation question). Higher levels correspond to utterances that fit in a broader dialog context (e.g. a place name after the system asks "Leaving from the airport. Is this correct?", or "HELP" in any context). Finally, non-understandings, which do not match any expectation, are given a matching level of $+\infty$.

Expectation level is strongly related to both finality and TIP duration. Pauses following partial utterances of expectation level 0 are significantly more likely to be turn boundaries than those matching any higher level. Also, very unexpected partial utterances (and non-understandings) contain shorter pauses than more expected ones. Another indicative feature for finality is the presence of a positive marker (i.e. a word like "YES" or "SURE") in the partial utterance. Utterances that contain such a marker are more likely to be finished than others. In contrast, the effect of negative markers is not significant. This can be explained by the fact that negative responses to confirmation often lead to longer corrective utterances more prone to pauses. Indeed, 91% of complete utterances that contain a positive marker are single-word, against 67% for negative markers.

Prosody

Prosody has traditionally be seen as a strong indicator of turn-taking phenomena. As reported in section 2.1.2, Duncan [1972], Oreström [1983], Chafe [1992], Ford and Thompson [1996], Koiso et al. [1998], Furo [2001] all found that certain prosodic patterns, such as falling or rising pitch or final vowel lengthening, foreshadowed turn transitions in human-human conversations. Ferrer et al. [2003] (see section 5.2) specifically found that vowel duration and pitch allowed them to improve automatic endpointing of user utterances.

Inspired by their results, we extracted three types of prosodic features: acoustic energy of the last vowel, pitch of the last voiced region, and duration of the last vowel. Vowel location and duration were estimated by performing phoneme alignment with the speech recognizer. Duration was normalized to account for both vowel and speaker identity. This is done by first dividing each vowel's duration by the mean duration of identical vowels in the whole corpus, to account for differences between vowels (e.g. the ”i” in ”bit” is much shorter than the ”oa” in ”float”). We then take the logarithm of the result and subtract the mean normalized duration of all vowels in the current dialog (so far), to account for speaker differences. Energy was computed as the log-transformed signal intensity on 10 ms frames. Pitch was extracted using the Snack toolkit [Sjolander, 2004], also at 10 ms intervals. For both energy and pitch, the slope of the contour was computed by linear regression, and the mean value was normalized by Z-transformation using statistics of the dialog-so-far. As a consequence, all threshold values for means are expressed in terms of standard deviations from the current speaker's mean value.

Vowel energy, both slope and mean, yielded the highest correlation with pause finality, although it did not rank as high as features from other categories. As expected, vowels immediately preceding turn boundaries tend to have lower and falling intensity, whereas rising intensity makes it more likely that the turn is not finished. On the other hand, extremely high pitch is a strong cue to longer pauses, but only happens in 5.6% of the pauses.

Timing

While they have often been overlooked in human-human studies, timing features, such as the duration-so-far of the current utterance, are intuitively likely to correlate with the likelihood that the turn ends. In addition, such features are easily extractable (though not necessarily perfectly reliable) in a running system, by simply keeping track of VAD and speech recognition events.

In fact, in our case, we found that timing features, provided the strongest cue to finality. The longer the on-going turn, the less likely it is that the current pause is a boundary. This is true both in terms of duration (i.e. the start time of the current pause relative to the beginning of the turn) and number of pauses observed so far. While this might seem like a paradox at first, it captures the fact that turns tend to be either very short (i.e. there is a high probability of the turn ending early on), or significantly longer.

The number of pauses so far also correlates well with mean TIP duration, the initial pauses of a turn tending to be longer than the later ones.

Speaker Characteristics

Because turn-taking behavior is to a large extent dictated by social conventions, cultural and individual differences affect participants' turn-taking behavior. Clancy et al. [1996] and Furo [2001] discuss differences in backchannels and turn transitions between English, Japanese, and Mandarin speakers. Within a culture, but across genders, Beattie [1982] found significant differences in the turn-taking behavior of British politicians during interviews.

To capture some of these individual differences, we consider the mean number of pauses per utterance, and the mean TIP duration observed so far for the current dialog. Both features correlate reasonably well with pause finality: a higher mean duration indicates that upcoming pauses are also less likely to be final, so does a higher mean number of pauses per turn.

Discussion

What emerges from the analysis above is that features from all aspects of dialog provide information on pause characteristics. While most previous research has focused on prosody as a cue to detect the end of utterances (e.g. Ferrer et al. [2003]), timing, discourse, semantic and previously observed pauses appear to correlate more strongly with pause finality in our corpus. While this can be partly explained by the specifics of the Let's Go system and the fact that prosodic features might be harder to reliably estimate on such noisy data than on commonly used research corpora, we believe this makes a strong case in favor of a broader approach to turn-taking for conversational agents, making the most of all the features that are readily available to such systems. Indeed, particularly in constrained systems like Let's Go, higher level features like discourse and semantics might be more robust to poor acoustic conditions than prosodic features. Still, our findings on mean TIP durations suggest that prosodic features might be best put to use when trying to predict pause duration, or whether a pause will occur or not. The key to more natural and responsive dialog systems lies in their ability to combine all these features in order to make prompt and robust turn-taking decisions. In the rest of this chapter, we describe our approach to go from feature analysis to decision making.

5.3.5 Performance of Supervised Classification

The most straight-forward way to detect turn boundaries is to directly classify pauses as final or internal using a standard classifier that uses the features available at the start of

the pause. The classifier can be trained in a supervised fashion on the collected data set. We trained a logistic regression model and evaluated it using 10-fold cross-validation on the Let's Go Winter '07 Corpus, using the exact same feature set as for our proposed approach (see 5.5.1). The model failed to improve over a majority baseline for classification accuracy (both with 14.5% error). On the other hand, log likelihood, a soft measure of accuracy, did improve from -0.41 to -0.35. This leads to two observations. First, the task of detecting turn boundaries among pauses is a difficult one, given the available features. Second, while the improvement in soft metric indicates that classifiers can be used to estimate the probability that the turn is finished at the start of a given pause, one cannot rely solely on classification to make the endpointing decision. This latter observation guided our design of an algorithm that combines classification and thresholding (see next section).

5.4 Dynamic Endpointing Threshold Decision Trees

5.4.1 Overview

Our goal is to build a decision tree where the internal nodes are questions on dialog features available at runtime at the start of a pause, and each leaf node contains an endpointing threshold to apply to this pause. The tree must be constructed so as to minimize the average latency (or average threshold value for turn boundaries) for a fixed false alarm rate. Because the thresholds themselves are unknown in our training set, we cannot rely on supervised approaches such as regression. Instead, we propose a two-stage algorithm that first builds a tree by clustering pauses according to an appropriate cost function and second, given the tree, finds the set of thresholds that minimizes latency for a given false alarm rate. The next sections provide a detailed description of both stages, as well as an analysis of the interaction between dialog features and pause distributions.

5.4.2 Feature-based Silence Clustering

The goal of the first stage of training is to cluster together pauses with a similar FA rate/latency trade-off. Specifically, we would like to generate low-threshold clusters, which contain mostly boundaries and short IPs, and clusters where long IPs would be concentrated and with very few boundaries, allowing to set high thresholds that reduce cut-in rate without hurting latency. To obtain such clusters, we designed a hierarchical algorithm that exhaustively searches binary splits by asking question on feature values such

as "Is the utterance duration more than 2000 ms?", "Is the expectation level of the latest partial recognition hypothesis smaller than 1?", or "Was the last system prompt an open question?". The question that yields the minimal overall cost is kept, where the cost C_n of cluster K_n is defined by the following function:

$$C_n = B_n \times \sqrt{\frac{1}{|K|} \sum_{p \in K} Duration(p)^2} \tag{5.1}$$

where B_n the number of TBs in K_n and $Duration(p)$ the duration of a pause p, set to zero for TBs. The intuition behind this formula is that the first factor captures the boundary ratio of the cluster and the second one, the root mean square of pause duration, is a variant of mean duration that puts a larger weight on long pauses. This process is repeated recursively for each generated cluster until the reduction in cost between the original cost and the sum of the costs of the two split clusters falls below a certain threshold. By minimizing $C(K)$, the clustering algorithm will find questions that yield clusters with either a small $|K_{TB}|$ (few TBs), or a small root mean square TIP duration (short TIPs), which, as we argued above, should lead to more appropriate thresholds. Ultimately, at the leaves of the tree are sets of pauses that will share the same threshold.

5.4.3 Cluster Threshold Optimization

Given the obtained clusters and under the hypothesis that TIP durations follow an exponential distribution (see Section 5.3.4), the unique set of threshold that minimizes the overall average latency for a certain total number of FAs, E is given by:

$$\theta_n = \frac{\mu_n \times log(\frac{E \times \mu_n}{\sum \mu_n}) + \mu_n \times log(\beta_n)}{B_n} \tag{5.2}$$

where μ_n and β_n can be estimated from the data.

An informal proof of 5.2 is as follows. Let (K_n) be a set of n pause clusters, the goal is to set the thresholds (θ_n) that minimize overall mean latency, while yielding a fixed, given number of false alarms E. let us define B_n the number of turn boundaries among the pauses of K_n. For each cluster, let us define $E_n(\theta_n)$ the number of false alarms yielded by threshold θ_n in cluster n, and the total latency L_n by:

$$L_n(\theta_n) = B_n \times \theta_n \tag{5.3}$$

The two curves in figure 5.5 represent L_n as a function of $E_n(\theta_n)$ for two hypothetical clusters. Assuming TIP durations follow an exponential distribution, as shown in Section

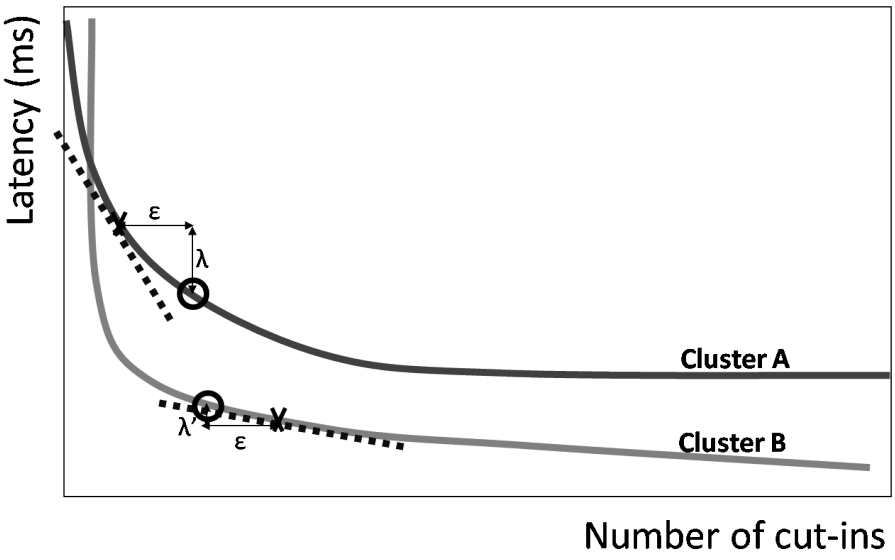


Figure 5.5: Illustration of the proof by contradiction of Theorem 1.

5.3, the following relation holds between L_n and E_n:

$$e^{\frac{L_n(\theta_n)}{\mu_n}} = \beta_n \times E_n(\theta_n) \tag{5.4}$$

where μ_K and β_K are cluster-specific coefficients estimated by linear regression in the log domain. If we take the log of both sides, we obtain:

$$L_n(\theta_n) = \mu_n \times log(E_n(\theta_n)) + \mu_n \times log(\beta_n) \tag{5.5}$$

Theorem 1. *If (θ_n) is a set of thresholds that minimizes $\sum_n L_n$ such that $\sum_n E_n(\theta_n) = E$, then $\exists A s.t. \forall n, \frac{dL_n}{dE_n}(\theta_n) = A$*

Proof. The proof can be done by contradiction, as illustrated in figure 5.5. Let us assume (θ_n) is a set of thresholds that minimizes $\sum_n L_n$, and $\exists (A, B) s.t. \frac{dL_A}{dE_A}(\theta_A) > \frac{dL_B}{dE_B}(\theta_B)$ (these thresholds correspond to the Xs on figure 5.5. Then, there exists small neighborhoods of θ_A and θ_B where $L_A(E_A)$ and $L_B(E_B)$ can be approximated by their tangents. Since the tangents' slopes differ, it is possible to find a small ϵ such that the increase in latency $\lambda = L_A(E_A - \epsilon) - L_A(E_A)$ is smaller than the decrease $\lambda' = L_B(E_B) - L_B(E_B + \epsilon)$

(these new thresholds correspond to the circles in figure 5.5). Since $E_A - \epsilon + E_B + \epsilon = E_A + E_B$, the total number of FAs is still equal to E but because $L_A(E_A - \epsilon) + L_B(E_B + \epsilon) < L_A(E_A) + L_B(E_B)$, $\sum_n L_n$ has been reduced, contradicting the hypothesis that (θ_n) minimizes $\sum_n L_n$. □

From Theorem 1, we get $\exists A s.t. \forall n \frac{dL_n}{dE_n} = A$. Thus, by deriving Equation 5.5, $\frac{\mu_n}{E_n} = A$ which gives $E_n = \frac{\mu_n}{A}$. Given that $\sum E_n = E$, $\frac{\sum \mu_n}{A} = E$. Hence, $A = \frac{\sum \mu_n}{E}$. From 5.5, we can infer the values of $L_n(\theta_n)$ and, using 5.3, the optimal threshold θ_n for each cluster given by 5.2.

5.5 Evaluation of Threshold Decision Trees

5.5.1 Offline Evaluation Set-Up

We evaluated our approach on the same corpus used for the analysis of Section 5.3. The feature set was extended to contain a total of 4 discourse features, 6 semantic features, 5 timing/turn-taking features, 43 prosodic features, and 6 speaker characteristic features. All evaluations were performed by 10-fold cross-validation on the corpus. Based on the proposed algorithm, we first built a decision tree and then computed optimal cluster thresholds for different overall FA rate. We report average latency as a function of the proportion of turns for which any TIP was erroneously endpointed, which is closer to real performance than pause FA rate since, once a turn has been endpointed, all subsequent pauses are irrelevant.

5.5.2 Overall Results

We compare the proposed approach to three baselines:

Fixed threshold a single pause duration threshold is used for all turns. The different cut-in rates are obtained for different global thresholds.

State-specific threshold A different threshold is used for each of the three dialog states (Open question, Closed question, Confirmation). For a given cut-in rate, the three thresholds are set so as to minimize the average latency, using 10-fold cross-validation. This is performed by running the proposed algorithm with dialog state as the only feature.

Ferrer et al (2003) We used the exact same set of features as for the proposed approach and trained classification trees using Matlab's `treefit` function. A different tree is trained for each set of pauses longer than 200 ms, 300 ms, 400 ms, 500 ms, 750 ms, and 1000 ms. All pauses that lasted 1200 ms or more were classified as end of turns. Given a set of input features corresponding to a pause, each tree gives a score (between 0 and 1), which we regularize using a sigmoid, thus obtaining the probability that the pause is the end of the turn. As noted in [Ferrer et al., 2003], the training corpus gets smaller for each increment of the threshold (e.g. there are fewer pauses longer than 300 ms than longer than 200 ms), and the accuracy of the classifier decreases. To compensate, we use a smoothed score T_i computed by interpolating the probability obtained from each tree T_i with the (interpolated) score of the previous tree $\hat{T}_{i-1}$. Thus $\hat{T}_1 = T_1$ and for all $i \geq 2$,

$$\hat{T}_i = \lambda T_i + (1 - \lambda)\hat{T}_{i-1}$$

with $\lambda = 0.7$[3]. At runtime, once a pause reaches a duration of 200, 300, 400 ms... the corresponding tree is run on the current features and a threshold θ is applied such that if $\hat{T}_i > \theta$, the system considers the pause as an end point, otherwise it waits until the next decision point. Varying the value of θ allows us to obtain different cut-in rates and draw the curve on Figure 5.6.

Figure 5.6 shows the performance of the four approaches. We give results on the 2-6% range of turn cut-in rate where any reasonable operational value is likely to lie (the 700 ms threshold of the baseline Let's Go system yields about 4% cut-in rate). The proposed approach significantly outperforms the fixed-threshold and state-specific-threshold baselines, up to a maximum latency reduction of, respectively, 25% and 13% for a 6% cut-in rate. The approach of Ferrer et al. [2003], however, tends to perform better for low cut-in rates (up to 3%) and identically for higher cut-in rates[4]. On the other hand, our approach generates a single, more compact, decision tree. The average number of internal nodes in cross-validation was 23.7 in our case (as can be seen at the far right of Figure 5.9), whereas Ferrer et al's approach results in several trees (one for each decision point, 6 in our case), with a total number of internal nodes of 44.

[3]This value was found empirically to yield the best result *on the test set*. Therefore the accuracy of this baseline is slightly inflated.

[4]Chapter 6 describes an approach improving over these results.

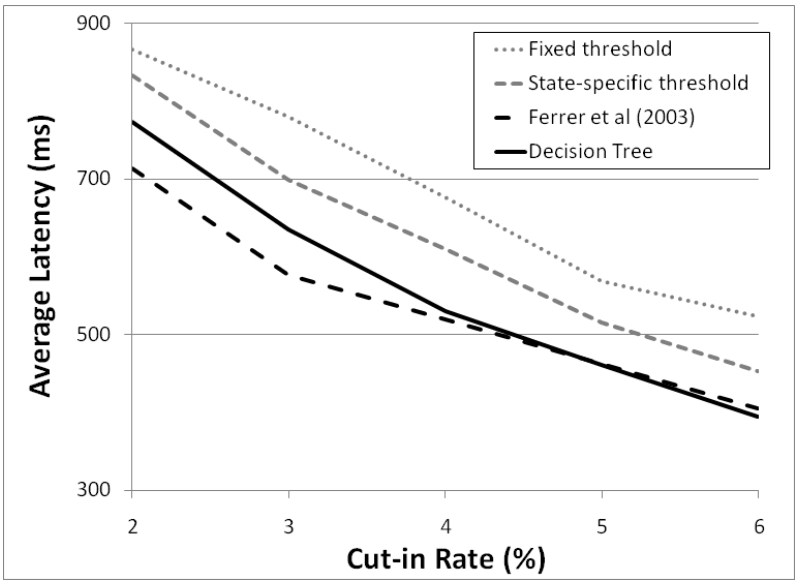


Figure 5.6: Performance of the proposed compared to a fixed-threshold baseline, a state-specific threshold baseline and the approach of Ferrer et al. [2003]. All confidence intervals for latency fall within $+/-4ms$. For the sake of clarity, we did not represent them on the graph.

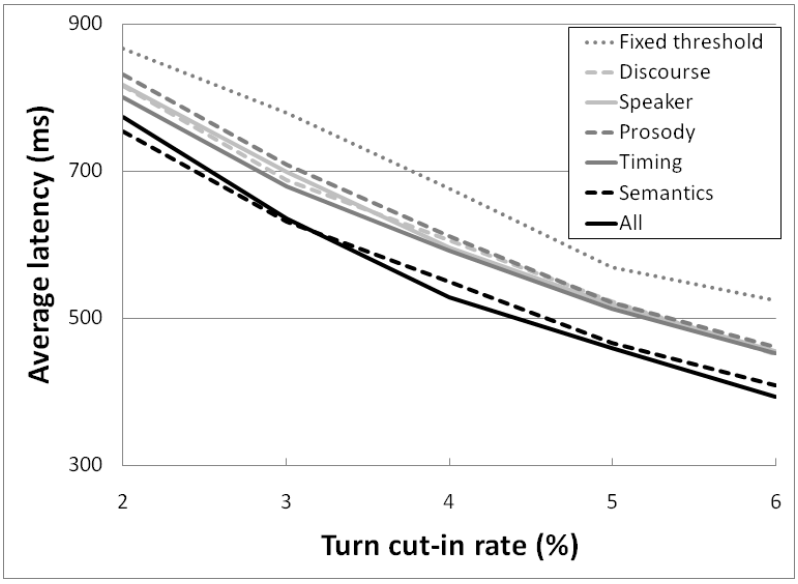


Figure 5.7: Performance of the proposed approach using different feature sets.

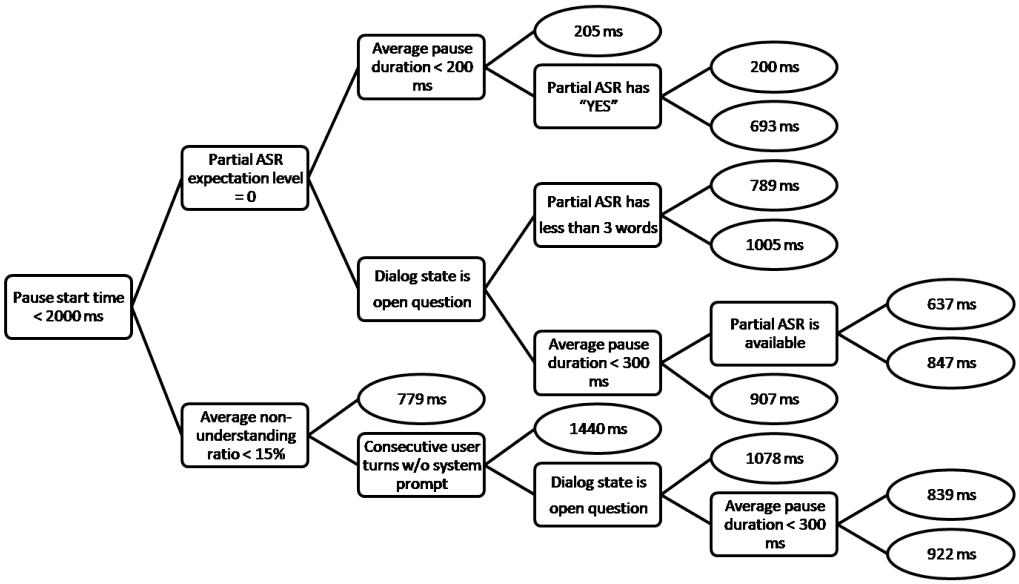


Figure 5.8: Example endpointing threshold decision tree learned by the proposed algorithm. Each internal node represents a test on dialog features. Cases for which the test is true follow the top branch while those for which it is not follow the bottom branch. Leaf nodes contain the thresholds obtained for a 3% overall cut-in rate.

5.5.3 Performance of Different Feature Sets

We evaluated the performance of each feature set individually. The corresponding cut-in rate/latency trade-off curves are shown in Figure 5.7. All feature sets improve over the fixed-threshold baseline. Statistical significance of the result was tested by performing a paired sign test on latencies for the whole dataset, comparing, for each cut-in rate the proportion of boundaries for which the proposed approach gives a shorter threshold than the single-threshold baseline. Latencies produced by the decision tree for all feature sets were all found to be significantly shorter ($p < 0.0001$) than the corresponding baseline threshold.

The best performing feature set is semantics, followed by timing, prosody, speaker, and discourse. The maximum relative latency reductions for each feature set range from 12% to 22%.

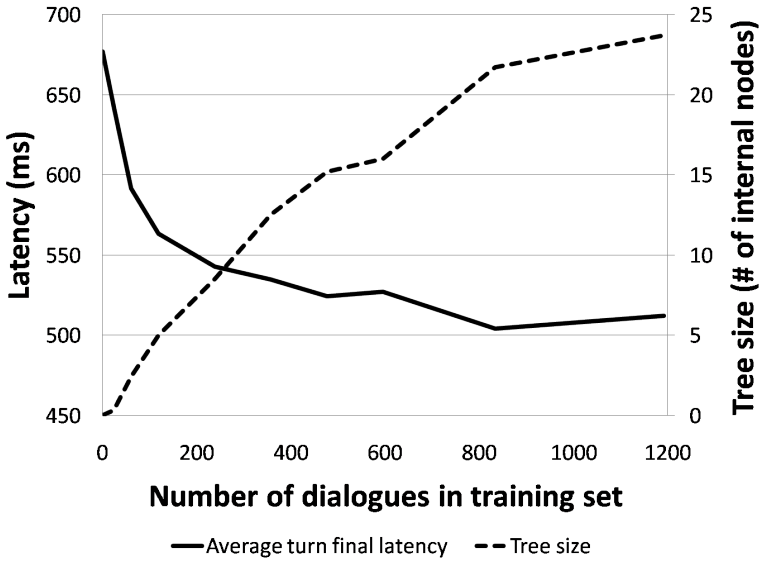
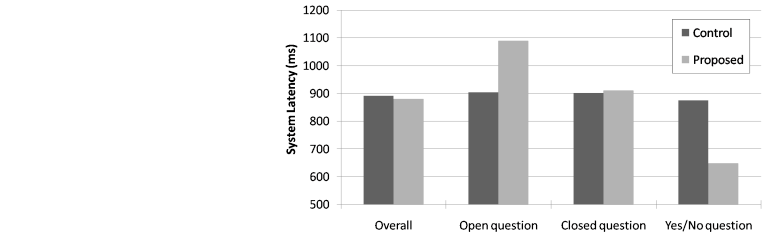


Figure 5.9: Performance and tree size with increasing training set size for a 4% cut-in rate.

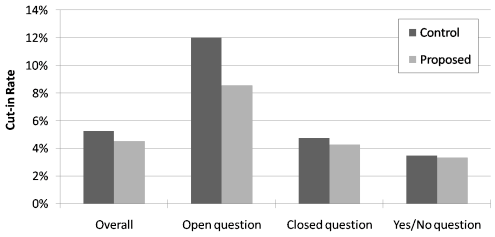
5.5.4 Learning Curve

In order to further analyze the performance of the algorithm, we computed its learning curve by progressively increasing the training set size to the total set of 1192 dialogs. We selected the thresholds corresponding to a 4% cut-in rate and performed, for each training set size, 10-fold cross-validation. The resulting mean performance is plotted on figure 5.9.

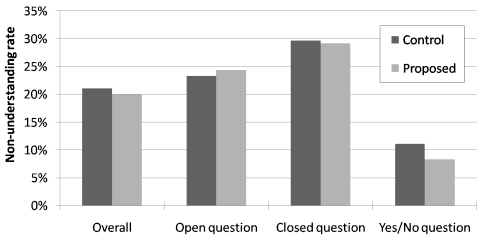
Most of the gain in average latency is obtained from the first 200-400 dialogs. Adding further dialogs to the training set increases tree size but only brings marginal improvement in performance. This indicates that, since the labeling of the training set is automatic (using the heuristic described in section 5.3.2), this algorithm could be used in an online fashion, constantly retraining the tree on the past few hundred dialogs. This would allow the system to adapt to any change in dialog structure (when adding a new subdialog), user population (e.g. seasonal variations, expert users adapting to the system), or any component of the system (e.g. new speech recognizer). It would only take a few hundred dialogs after any change for the system to automatically recompute optimal endpointing thresholds.



(a) Latency



(b) Cut-in rates



(c) Non-understanding rate

Figure 5.10: Live evaluation results.

5.5.5 Live Evaluation

We confirmed the offline evaluation's findings by implementing the proposed approach in Let's Go's Interaction Manager. Since prosodic features were not found to be helpful and since extracting them online reliably (in particular, pitch) is typically resource-expensive, we did not include them. At the beginning of each dialog, the system was randomly set as a baseline version, using a 700 ms fixed threshold, or experimental version using the tree learned from the offline corpus represented in figure 5.8 with thresholds corresponding to a theoretical cut-in rate of 3%. We collected a total of 1061 dialogs in May 2008, 548 in the control condition and 513 using the proposed approach.

Figure 5.10 shows the results. Overall latency (see figure 5.10(a)), which includes not only the threshold but also the time it took for the system to generate and synthesize the response, was almost the same between the two conditions (although the difference in median value, as measure by a Wilcoxon rank sum test, is statistically significant, with $p < 1e - 8$). However, larger differences appear when examining the three main contexts separately. The proposed approach was in fact slower than the baseline after open questions ($p < 1e - 8$). The corresponding user utterances are often long and contain hes-

itations, hence longer internal pauses. The proposed approach is also slower, though only very slightly, after closed questions ($p < 1e-8$). The difference is reversed after yes/no questions, which mostly yield one-word answers with no internal pause ($p < 1e-8$). As expected, the algorithm learned to set longer thresholds for the former and shorter thresholds for the latter. The effect of this can be seen on cut-in rates (figure 5.10(b)). Overall, the cut-in rate in the experimental condition is 4.5%[5] vs 5.3% in the control condition, which is statistically significant ($p < 0.05$, with a binomial proportions test). Not surprisingly, the bulk of the gain comes from responses to open questions ($p < 0.01$), thanks to the longer thresholds. The differences observed in the other two states are not statistically significant. The very short thresholds used after yes/no questions did not harm cut-in rate, again because these utterances contain very few internal pauses. However, because this approach is contingent on pause detection, which by itself requires a certain amount of audio data to make a decision, the latency induced is still fairly high (about 650 ms). We will see in Chapter 6 how we can perform endpointing without relying on pauses in unambiguous cases.

Finally, to get more insight on the impact of endpointing on speech recognition performance, we analyzed non-understanding rate (figure 5.10(c)). Overall, the proportion of non-understandings was reduced by 21.1% for the baseline to 20.0% for the proposed approach, which appears as a trend in our data ($p < 0.1$). When looking at specific contexts, the only statistically significant difference is after yes/no questions, where the rate went from 11.1% to 8.3% ($p < 1e-4$). This result was not expected since cut-in rates were virtually identical in that context. However, we have already discussed a similar phenomenon in section 5.3.3. Longer thresholds increase the risk of background noise hurting recognition performance. Therefore, the fact that the algorithm significantly shortens threshold in this context improved not only responsiveness but also speech recognition performance.

[5]The difference between the theoretical cut-in rate of 3% and the observed cut-in rate of 4.5% can be explained by the imperfection of the automatic labeling heuristic (see section 5.3.2) as well as by differences in user behavior (in particular, we assumed in the batch experiment that once a turn was cut-in, the user would stop and let the system speak, which might not have been the case in reality, potentially leading to several cut-ins in a row).

Chapter 6

The Finite-State Turn-Taking Machine

6.1 Summary

In this chapter, we propose a comprehensive probabilistic model of turn-taking to control the Interaction Manager introduced in Chapter 4. In section 6.2, we review existing finite-state models of turn-taking in dyadic conversations, and propose the Finite-State Turn-Taking Machine, an approach to turn-taking that relies on three core elements:

- A non-deterministic finite-state machine that captures the state of the conversational floor
- A cost matrix that models the impact of different system actions in different states
- A decision-theoretic action selection mechanism through which the system base its turn-taking decisions

The next three sections explain the application of the FSTTM to two key turn-taking phenomena. First, we revisit the problem of end-of-turn detection at pauses described in Chapter 5 in section 6.3. In section 6.4, we generalize the approach to allow the system to take turns even when a pause has mostly started but has not yet been detected. Finally, section 6.5 focuses on the problem of detecting user interruptions (aka barge-in) during system prompts. Drawing from the work presented in Chapter 5, we illustrate, in each case, how a wide range of dialog features can be exploited to estimate state probabilities. Evaluation both offline and by applying the FSTTM to the live Let's Go system showed that it outperforms previously proposed data-driven approaches, including the one intro-

duced in Chapter 5. We discuss the flexibility of the model, its strengths and possible applications and extensions.

6.2 Turn-Taking States and Actions

6.2.1 Conversational Floor as a Finite-State Machine

6-state finite state models of turn-taking

In the 1960's and early 1970's, several researchers proposed models to explain the rhythmic turn-taking patterns in human conversation. In particular, Jaffe and Feldstein [1970] developed a fully automated process to extract temporal patterns of dyadic conversations. Their system (the Automatic Vocal Transaction Analyzer) generates traces of who is speaking within 300 ms frames of conversations recorded with the device. With two participants (A and B), there are four possible configurations at any point in time: A and B both speak, A and B are both silent, only A speaks, and only B speaks. Jaffe and Feldstein use these traces to compute the mean durations of pauses, switching pauses (when a different speaker takes the floor), simultaneous speech, and (single-speaker) vocalizations. Based on these traces, they found that the duration of each of these three types of event follow exponential distributions. This finding led them to propose first-order Markov models to capture the alternation of speech and silence first in monologues, then in dialog (first-order Markov processes generate exponential distributions for the duration of each state). They find that their initial four-state model (corresponding to the four configurations mentioned above) fails to distinguish between silences that happen immediately following A's speech and those following B's speech. A similar limitation exists for overlapping speech. Based on this observation, they extend their model to one similar to the six-state model shown in Figure 6.1. This model explicitly accounts for who has the floor at every point in time, in addition to who is vocalizing. Jaffe and Feldstein found this model to better fit their data than the four-state model. At the same time as Jaffe and Feldstein, and while concerned more with electrical engineering and telecommunications rather than medical or psychological applications, Brady Brady [1969] developed a very similar six-state model. He then trained the parameters of the model (i.e. the state transition probabilities) on the recordings of 16 dyadic conversations, which were annotated for speech/non-speech for each participant at a 5 ms time frame. Speech was detected frame-by-frame automatically and then smoothed by filling in gaps of less than 200 ms and rejecting speech periods of less than 15 ms. The approach was evaluated by training the parameters on a given conversation and generating a simulated conversation with the model, using a Monte Carlo

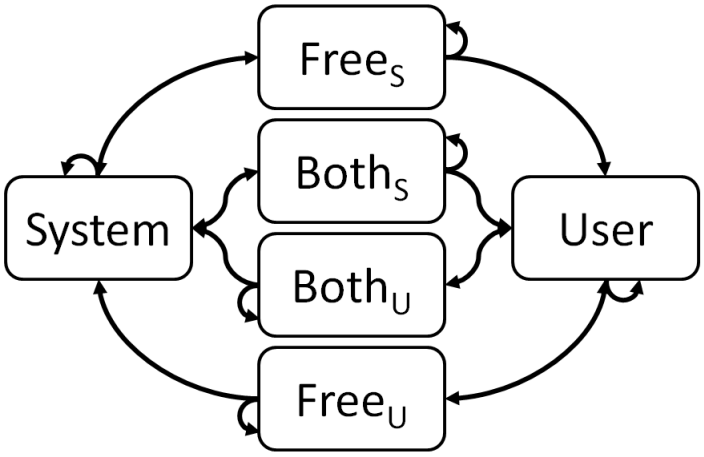


Figure 6.1: Our six-state model of turn-taking, inspired by Jaffe and Feldstein [1970] and Brady [1969].

simulation. Then the generated conversation was compared to the original real one along several dimensions (pause and speech segment durations, overlaps, etc). Brady found that his model produced a generally good fit of the data. One interesting exception is that of the distribution of the amount of time between the beginning of a speech interval (called talkspurt by Brady) and the beginning of an interruption by the other participant. Real conversations exhibited significantly more early interruptions (i.e. short intervals) than the model would predict. Brady explains this with the fact that these ”early” interruptions are probably not intended to be interruptions but rather the result of both speakers starting to speak inadvertently at the same time (both assuming they would be the only speaker). Whereas the model assumes that as soon as a participant starts speaking, the other one is aware of it and will only speak if she wishes to interrupt, in reality, it takes some amount of time (Brady suggests 200 ms) for a participant to register the fact that the other has started speaking. As we will see, we similarly observed the preponderance of early (as well as late) user barge-ins in our data.

Finite-State Models for Control

While Jaffe and Feldstein [1970] and Brady [1969] are primarily concerned with the analysis of human-human conversations, more recently, several researchers have proposed finite-state machines to control conversational agents. For instance, Cassell et al. [2001] models the conversational state of an embodied real estate agent as a 5-state machine. Two states indicate whether a user is present or not, whereas the other three indicate who holds the floor between the user and the agent, or whether the floor is open. The agent tracks the current conversational state through a number of sensors (visual and auditory), so that the user can claim the floor by either speaking or gesturing to the agent. The agent can also hold the floor while she is planning a response, by looking away from the user. Floor conflicts are not captured by this machine and are presumably resolved through simple rules (e.g. when the user speaks, the agent immediately yields the floor).

From the turn-taking models already described in Chapter 2, Kronild [2006] proposes a much more complex model, based on Harel statecharts, which are an extension of finite-state machines for modeling and visualizing abstract control Harel [1987]. This model takes into account any number of participants and also captures projections, a key aspect of the SSJ model of turn-taking[Sacks et al., 1974]. Harel statecharts feature hierarchical states, which allow this model to capture both broad characteristics of conversation (e.g. ”I am speaking” vs ”I am silent”) and fine-grain aspects such as whether an overlap with another speaker is occurring or even whether a TRP in a speaker's utterance is predicted.

Thorisson [2002] also features an implicit finite-state machine, in the form of ”State Deciders” which are rules governing the transitions between dialog states such as ”Other-Has-Turn” and ”I-Take-Turn”.

All these models are deterministic. At any point in time, the agent *knows* the state of the conversational floor and uses fixed, hand-written, rules to take appropriate actions. These approaches assume 1) that sensors provide reliable information on the state of the world, and 2) that the state itself is unambiguous. These assumptions are made explicit in Thorisson's hypothesis that:

> A decision is based on the boolean combination of perceptual features.

6.2.2 Overview of the Finite-State Turn-Taking Machine

Extending the 6-state model to control

Our model, the Finite-State Turn-Taking Machine (FSTTM), uses the same six states as Jaffe and Feldstein [1970] and Brady [1969]. However, we apply this model to the control of a conversational agent, with a goal similar to that of Cassell et al. [2001], Kronild [2006], and Thorisson [2002]. One important contribution is that we define the states in terms of the participants' *intentions* and *obligations* rather than the surface level observation of speech vs silence (see our discussion of the definition of floor in section 1.2). For example, the state is $USER$ when the user has the obligation to speak (to respond to a system question) or the intention to speak, while at the same time, the system does not hold the floor. This does not necessarily means that the user is speaking, for example at pauses during a user utterance or right before the user starts an utterance.

As can be seen in Figure 6.1, not all transitions are valid. First, there is no direct transition between any of the intermediate states (the two $FREE$ states and two $BOTH$ states). The assumption is that to go from any of these state to another, the model will first go to either $SYSTEM$ or $USER$. This is an approximation as there might be cases where, for example, both the system and user start speaking at the exact same time, going from a $FREE$ state to a $BOTH$ state. However these cases are rare enough that they can be approximated using a transition through either $SYSTEM$ or $USER$. Second, because intermediate states are conditioned on who had the floor previously, not all valid transitions are bidirectional. For example, there is no transition from $SYSTEM$ to $BOTH_U$. We associate pairs of user/system actions to each transition. The four possible actions are:

- **G**rab the floor
- **R**elease the floor
- **W**ait while not claiming the floor
- **K**eep the floor

For example, transition from $SYSTEM$ to $FREE_S$ corresponds to the user waiting silently and the system releasing the floor at the end of a prompt, noted (R, W) (we always note the system action first and user action second).

This representation allows us to formalize a wide variety of turn-taking phenomena in a unified framework. Specifically, there are 4 types of 2-step transitions from a single

-floor-holder state ($SYSTEM$ or $USER$) to another (or the same) single-floor-holder state, which represent typical turn-taking phenomena:

Turn transitions with gap are the most common way the floor goes from one participant to the other. For example, starting with a user utterance, once the user finishes speaking, the floor becomes free, after which the system starts responding, thus grabbing the floor. The resulting state sequence, aligned with the first two turns of the example first given in section 4.2, is:

$$USER \overset{(W,R)}{\rightarrow} \quad FREE_U \overset{(G,W)}{\rightarrow} \quad SYSTEM$$

User: I want to go to Miami.

System: Going to Miami, right?

Conversely, the transition with gap following a system prompt corresponds to:

$$USER \overset{(R,W)}{\rightarrow} FREE_S \overset{(W,G}{\rightarrow} USER$$

Turn transitions with overlap happen when a participant grabs the floor while it still belongs to the other. For example, when a user barges in on a system prompt, both participants hold the floor. Then, the system recognizes the barge-in attempt and relinquishes the floor, which becomes user's.

$$SYSTEM \overset{(K,G)}{\rightarrow} BOTH_S \overset{(R,K}{\rightarrow} USER$$

And conversely, when the system interrupts the user mid-utterance (which in dialog systems is more often the result of an intentional cut-in, rather than intentional interruption), the state sequence is:

$$USER \overset{(G,K)}{\rightarrow} BOTH_U \overset{(K,R)}{\rightarrow} SYSTEM$$

Failed interruptions happen when a participant barges in on the other and then withdraws before the original floor holder releases the floor. For example, when the system interrupts the user (often by mistake) but detects it and interrupts itself:

$$USER \overset{(G,K)}{\rightarrow} BOTH_U \overset{(R,K}{\rightarrow} USER$$

The converse is usually the result of the system failing to react fast enough to a user barge-in:

$$SYSTEM \overset{(K,G)}{\rightarrow} BOTH_S \overset{(K,R)}{\rightarrow} SYSTEM$$

Note that backchannels seem to fit in this category too. However, since backchannels, by definition, do not represent an attempt to grab the floor, they are not captured by the model as it is (for example, the floor should remain $SYSTEM$ when a user backchannels a system utterance).

Time outs start like transitions with gap but the intended next speaker (e.g. the user after a system prompt) does not take the floor and the original floor holder grabs it back. For instance, after a system prompt, if the floor remains free for a certain amount of time, the system attempts to re-establish the communication with the user, as follows:

$$SYSTEM \overset{(R,W)}{\rightarrow} FREE_S \overset{(G,W}{\rightarrow} SYSTEM$$

The opposite also happens when the system is to slow to respond to the user:

$$USER \overset{(W,R)}{\rightarrow} FREE_U \overset{(W,G}{\rightarrow} USER$$

While all the transitions above were described as deterministic, the actual state of the model is not fully observable. Specifically, while the system *knows* whether its claiming the floor or not, it can only *believe* with some degree of uncertainty that the user does so. The system's knowledge of its own claim to the floor splits the state space into two disjoint subsets. When the system claims the floor, the state can be $SYSTEM$, $BOTH_S$, or $BOTH_U$. When the system does not claim the floor, the state can be $USER$, $FREE_U$, or $FREE_S$. In either case, the system needs to recognize the user's intention (i.e. whether the user claims to the floor or not) to maintain a probability distribution over the three states. Since the distinction between the two $BOTH$ states (resp. the two $FREE$ states) is based on past history that can be known with a high level of certainty, the uncertainty in state distribution is fully characterized by the probability that the user is claiming the floor, which will have to be estimated from observations, as we will see below.

6.2.3 Cost of Turn-Taking Actions

The problem we are facing is that of choosing the best system action given the system's belief about the current state of the model. That is achieved by applying the probabilistic decision theory principle of selecting the action with lowest expected cost. The actions available to the system are the four described above (G,R,K,W), although not all actions are available in all states. In fact, as can be seen in Table 6.1, there are always only two actions available in each state, depending on whether the system is claiming the floor or not.

State \ Action	K	R	W	G
$SYSTEM$	0	C_S	-	-
$BOTH_S$	C_O	0	-	-
$BOTH_U$	C_O	0	-	-
$USER$	-	-	0	C_U
$FREE_U$	-	-	$C_G \cdot t$	0
$FREE_S$	-	-	$C_G \cdot t$	0

Table 6.1: Cost of each action in each state (K: keep the floor, R: release the floor, W: wait without the floor, G: grab the floor, t: time spent in current state, -: action unavailable).

Each action in each state has a particular cost. While there are many possible ways of defining these costs, we propose a simple cost structure that derives from the principles laid out in Sacks et al. [1974]:

> Participants in a conversation attempt to minimize gaps and overlaps.

Critics of Sacks et al's theory have pointed our that this principle fails to capture phenomena such as backchannels and collaborative endings [Clark, 1996]. However, while this criticism holds with Sacks et al's definition of "gaps" and "overlaps" as silence and simultaneous speech, our definition of the floor at the intention level addresses most of these concerns. For example, backchannels are, by definition, not attempts at grabbing the floor, and thus do not constitute overlaps by our definition. From this general principle, we derive three rules to drive the design of the cost matrix:

1. The cost of an action that resolves either a gap or an overlap is zero
2. The cost of an action that creates unwanted gap or overlap is equal to a constant parameter (potentially different for each action/state pair)
3. The cost of an action that maintains a gap or overlap is either a constant or proportional to the total time spent in that state

The resulting cost matrix is shown in Table 6.1, where

- $\boldsymbol{C_S}$ is the cost of interrupting a system prompt before its end when the user is not claiming the floor (false interruption)
- $\boldsymbol{C_O}$ is the cost of remaining in overlap
- $\boldsymbol{C_U}$ is the cost of grabbing the floor when the user is holding it (cut-in)
- $\boldsymbol{C_G}$ is the cost of remaining in gap

This cost structure makes a number of simplifying assumptions. First, in reality, the cost of creating a gap or overlap is not necessarily constant. For example, the cost of interrupting the user might vary depending on what has already been said in the utterance, so does the cost of interrupting a system prompt. Second, for C_G and C_O we tried three cost functions: constant, sublinear (log) with time, and linear with time and selected the ones that gave the best results. A more principled approach to setting the costs would be to estimate from perceptual experiments or user studies what the impact of remaining in gap or overlap is compared to that of a cut-in or false interruption. However, as a first approximation, the proposed cost structure offers a simple way to take into account some of the constraints of interaction. We will discuss potential improvements to the cost matrix in the future work section.

6.2.4 Decision Theoretic Action Selection

Given the state space and the cost matrix given above, the optimal decision at any point in time is the one that yields the lowest expected cost, where the expected cost of action A is:

$$C(A) = \sum_{S \in \Sigma} P(s = S|O) \cdot C(A, S)$$

where Σ is the set of states, O are the observable features of the world, and $C(A, S)$ is the cost of action A in state S, from the cost matrix in Table 6.1. In addition to the cost matrix' four constants, which we will consider as parameters of the model, it is thus necessary to estimate $P(s = S|O)$, which as seen above amounts to estimate the probability that the user is claiming the floor. The way to estimate this probability varies depending on the circumstances. In the following two sections, we discuss the two main turn-taking phenomena encountered in typical dialog systems, endpointing (or end-of-turn detection) and user interruption (or barge-in) detection.

6.3 Pause-based Endpointing with the FSTTM

6.3.1 Problem Definition

In the FSTTM formalism, endpointing is the problem of selecting between the Wait and the Grab actions during a user utterance. At each pause within the utterance, uncertainty arises as to whether the user is keeping the floor, in which case the state is $USER$, or releasing it, in which case the becomes $FREE_U$[1].

This first approach to endpointing is similar to that described in Chapter 5 in that, whenever user speech is detected by the voice activity detector, the probability of being in state $USER$ is set to 1. Only when a pause is detected by the VAD does the system attempt to estimate $P(s = USER|O_t) = 1 - P(s = FREE_U|O_t)$, where O_t represents the set of observable features characterizing a pause that has lasted t milliseconds so far. We further decompose O_t into t and O, which is the set of dialog features available at the time when the pause started (i.e. at $t = 0$), under the assumption that once the pause is started, no additional information, apart from the duration of the pause arises. Dialog features are identical to those described in Chapter 5, and cover discourse, semantics, timing, and user characteristics. We decided not to include prosody after early attempts that showed that, as was the case in Chapter 5, it did not significantly affect the performance of the model.

We express the probability that the state is $USER$, in a pause that has lasted t milliseconds so far, as follows (NB: for the sake of clarity, we note $s = USER$ as U):

$$P(s = USER|O_t) = P(U|O, t) \tag{6.1}$$

$$= \frac{P(U, t|O)}{P(t|O)} \tag{6.2}$$

$$= \frac{P(t|O, U) \cdot P(U|O)}{P(t|O)} \tag{6.3}$$

where $P(t|O, U)$ is the probability that a pause lasts at least t milliseconds, given that the user still claims the floor and given the dialog features at the start of the pause, $P(U|O)$ is the probability that the user still claims the floor (i.e. that this pause is turn-internal rather than turn-final) given the dialog features (regardless of how long the pause lasts), and $P(t|O)$ is the probability that the pause lasts at least t milliseconds.

[1]We make the assumption that, at some point during this utterance before the first pause, we know with probability 1 that the floor was $USER$. This assumption means that we do not attempt to model here those cases where the VAD wrongly detects speech and the user never really take a turn. We will focus more on these issues when dealing with interruption detection in section 6.5.

The probability that the state is $FREE_U$ (i.e. that the user released the floor) can be similarly decomposed as:

$$P(s = FREE_U|O_t) = \frac{P(t|O,F) \cdot P(F|O)}{P(t|O)} \quad (6.4)$$

$$= \frac{1 \cdot (1 - P(U|O))}{P(t|O)} \quad (6.5)$$

$$= \frac{1 - P(U|O)}{P(t|O)} \quad (6.6)$$

Equation 6.5 uses the fact that the user never resumes speaking in the $FREE_U$ state (hence $P(t|O,F) = 1$). Note that overall, we work under the assumption that once the user has released the floor to $FREE_U$, they never claim it back (see the "time out" phenomena in section 6.2.2). While this is an approximation, we can assume and have actually observed in the data that the time frame at which the user would in fact resume speaking is significantly longer than the longest "normal" system reaction time [2]. Therefore, this assumption does not hurt the decision we are making for endpointing.

Given Table 6.1, the expected cost of **W**aiting is:

$$C(W,t,O) = P(U|t,O) \cdot C(W,U) + P(F|t,O) \cdot C(W,F) \quad (6.7)$$

$$= P(U|t,O) \cdot 0 + P(F|t,O) \cdot C_G \cdot t \quad (6.8)$$

$$= \frac{P(F|O)}{P(t|O)} \cdot C_G \cdot t \quad (6.9)$$

and the expected cost of **G**rabbing the floor is:

$$C(G,t,O) = P(U|t,O) \cdot C(G,U) + P(F|t,O) \cdot C(G,F) \quad (6.10)$$

$$= P(U|t,O) \cdot C_U + P(F|t,O) \cdot 0 \quad (6.11)$$

$$= \frac{P(t|O,U) \cdot (1 - P(F|O))}{P(t|O)} \cdot C_O \quad (6.12)$$

At $t = 0$, i.e. at the very beginning of the pause, $C(W,t,O) = 0$ and, except for the degenerate case where either $P(F|O) = 1$ or $P(t|O,U) = 0$, $C(G,t,O) > 0$. As t grows, $C(W,t,O)$ increases while $C(G,t,O)$ decreases (because typically $P(t|O,U)$ decreases faster than $P(t|O)$ with time. So there is necessarily a time t_0 so that $C(W,t_0,O) =$

[2] We still observe user time outs when, for example, the voice activity detector fails to detect silence at the end of the user turn and the system takes more than a few seconds to respond but these cases do not interfere with genuine attempts at endpointing because, for practical reasons, we force endpointing when a pause has lasted more than 1500 ms.

$C(G, t_0, O)$. This time corresponds in fact to the point at which the cost of waiting becomes higher than that of grabbing the floor; in other words it is an endpointing threshold similar to those described in Chapter 5. Given all of the above, the threshold t_0 is the solution to the equation:

$$C(W, t_0, O) = C(G, t_0, O) \tag{6.13}$$

$$\frac{P(F|O)}{P(t|O)} \cdot C_G \cdot t = \frac{P(t|O,U) \cdot (1 - P(F|O))}{P(t|O)} \cdot C_O \tag{6.14}$$

$$P(F|O) \cdot C_G \cdot t = P(t|O,U) \cdot (1 - P(F|O)) \cdot C_O \tag{6.15}$$

$$t = \frac{C_O}{C_G} \cdot P(t|O,U) \cdot \frac{1 - P(F|O)}{P(F|O)} \tag{6.16}$$

We define $\alpha = \frac{C_O}{C_G}$ as a parameter of the model. To solve the above equation, we need estimates of both $P(F|O)$ and $P(t|O,U)$, which are described in the next two sections. Note that $P(t|O)$, which appeared on both sides of equation 6.14 can be eliminated and thus does not need to be estimated.

6.3.2 Estimating $P(F|O)$

We estimate $P(F|O)$ by stepwise logistic regression on a training set of pauses labeled for finality (whether the pause is turn-final or turn-internal) with their associate dialog features automatically extracted as in Chapter 5. We created one additional feature to capture lexical cues that mark the end of turns. First we collected all partial and final hypotheses collected by the live system in the month of April 2008. This resulted in a corpus of 133828 hypotheses. We marked the end of each partial hypothesis by a special token (</p>) and that of final hypotheses with a different one (</s>). We trained a standard 3-gram language model for each of the three main dialog states (open question, closed question, and confirmation, see section 5.3.2) on this corpus using the CMU-Cambridge Statistical Language Model Toolkit [Clarkson and Rosenfeld, 1997]. Given the sequence of words of a hypothesis, we use the generated LM to compute the log-likelihood of both the partial hypothesis symbol and the final hypothesis symbol given the last two words of the hypothesis. Finally, we define our new feature as the difference of these two log-likelihoods (i.e. the log of the likelihood ratio). This feature, which takes values between approximately -6 and $+2$ is larger for hypotheses whose last words match existing final hypotheses and smaller for those that do not. Inspection of the produced LM shows that it captures expected phenomena such as the fact that the single-word hypothesis "YES" is likely to be final in the confirmation state. It also learns less expected patterns. For example, among the bus routes that are covered by the system, which include routes numbered

Feature	Value range	Median	Open	Closed	Confirmation
Constant	-	-	1.5409	2.1298	1.2403
LM boundary score	$[-3.9, 2.1]$	-0.3	0.6681	1.1009	0.6541
Avg nb words per turn	$[1, 27]$	1.9	-0.2723	-0.2296	0
Hyp has confirm	$0, 1$	0	0	0	1.0732
User barge-in	$0, 1$	0	0.8447	0.4829	0
Avg confidence so far	$[0, 1]$	0.6	0	0	1.5474
First agenda level	$[0, 7] \bigcup 999$	1	0	0	-0.0006

Table 6.2: Features selected by stepwise logistic regression to estimate $P(F|O)$ at pauses and their coefficients (all non-null coefficients are non null with $p < 0.01$).

61A, 61B, and 61C, 61A is less likely to be final than the others. This is because routes that start with the number 61 tend to be recognized as 61A in the early partial hypotheses. Thus seeing 61A appear in a hypothesis is not as strong a sign that the turn is finished as seeing, for example, 61C.

Based on these features, we built one logistic regression model for each dialog state using stepwise regression and 10-fold cross-validation on the data used in Chapter 5. Table 6.3 shows the performance of the model for each state, as well as overall. The learning algorithm is actually unable to improve the classification error rate, although the probability estimate gets more accurate as is evidenced by the increase in log likelihood (all likelihood ratio test show statistically significant differences at the $p < 1e-4$ level). The features selected for each state by the stepwise algorithm are shown in Table 6.2. The LM boundary score is the only feature used in all three states and also the one that, when normalized by its dynamic range, has the largest weight, showing its general informativeness. The open and closed question states use the same other features, which consists of the average number of words per turn in the dialog so far and a boolean feature indicating whether the user is barging in on the system. The first feature shows that users who have had longer turns so far are also more likely to have internal pauses (correlated with longer turns) in the current turn. The second one indicates that barge-ins tend to have fewer internal pauses than other utterances. Three other features are helpful for the confirmation state. Whether the partial hypothesis contains a confirmation word (e.g. "YES") is strongly correlated with high $P(F|O)$ since many utterances in this state are single "YES". Next, the mean turn confidence score so far captures how well speech recognition has been working so far, and also how likely a user is to answer "YES" to an explicit confirmation request. Finally, the level

Dialog state	Majority baseline	Classification error rate	Baseline log likelihood	Log likelihood
Open question	37.91%	34.71%	-0.6636	-0.6073
Closed question	24.94%	25.50%	-0.5616	-0.4973
Confirmation	11.66%	12.02%	-0.3601	-0.2963
Overall	21.88%	21.72%	-0.5254	-0.4375

Table 6.3: Performance of state-specific logistic regression for estimating $P(F|O)$ at pauses.

at which the current partial hypothesis matches the system's expectations mostly captures non-understandings (non-understandings get an arbitrary 999 value, whereas matches get actual agenda levels between 0 and 6. See [Bohus and Rudnicky, 2003] for an explanation of the agenda).

6.3.3 Estimating $P(t|O,U)$

$P(t|O,U)$, the probability that the current pause will last at least t milliseconds given that it is a turn-internal pause, is directly related to the distribution of pause durations. It plays a very similar role to the duration component of the clustering cost function introduced in Section 5.4.2. Here again, we exploit the observation made in Section 5.3.3 that pause durations approximately follow an exponential distribution:

$$P(t|O,U) \quad = \quad e^{-\frac{t}{\mu(O)}} \tag{6.17}$$

where μ is the only parameter of the distribution, which is equal to the mean pause duration. To estimate $\mu(O)$ we train a regression model using the dialog features O to predict the duration of turn-internal pauses. Because duration are lower bounded by zero, we use a generalized linear model[McCullagh and Nelder, 1989] with a gamma distribution (exponential distributions are a special case of gamma distributions) and a logarithmic link. In addition to improving the fit, the logarithmic link guarantees that the predicted durations are positive.

Predicting the duration of a pause at its onset is a difficult task and the fit is far from perfect. The correlation between actual and predicted duration is 0.43 and the root mean square error 248 ms. The first four feature picked by the stepwise algorithm are user

barge-in (YES/NO), dialog state (last system prompt type), whether a partial recognition hypothesis is available (YES/NO), the number of pauses since the beginning of the utterance, and the time since the end of the last user utterance.

For each pause, we compute the predicted pause duration and use that duration as the μ parameter of $P(t|O,U)$.

6.3.4 Batch Evaluation

We evaluated our FSTTM approach to endpointing at pauses on the dataset already used for clustering-based threshold optimization in Chapter 5. Figure 6.2 shows the latency / cut-in trade-off for the fixed threshold baseline, threshold optimization, the approach proposed by Ferrer et al. [2003] (see section 5.5.2 for details) and the FSTTM approach. The FSTTM consistently outperforms all other approaches, improving over the fixed threshold baseline by up to 29.5%.

In order to assess the impact of $P(t|O,U)$, we tested three different ways of computing the μ parameter of the exponential distribution:

1. Overall mean pause duration, independently of the dialog features
2. Pause duration predicted by the generalized linear model described in the previous section
3. An oracle that predicts the actual pause duration for all internal pause and the overall mean for all final pauses

As seen in Figure 6.3, predicting pause duration provides almost no improvement over a fixed pause duration. However, the oracle duration did provide significant improvement, which confirms the potential of using an estimated $P(t|O,U)$, provided better modeling of pause duration can be achieved (e.g. using better prosodic features).

6.4 Anytime Endpointing

6.4.1 Problem Definition

While the approach described above allowed some improvement over the method of Chapter 5, it is still constrained by the pause detection mechanism, which, independently of the

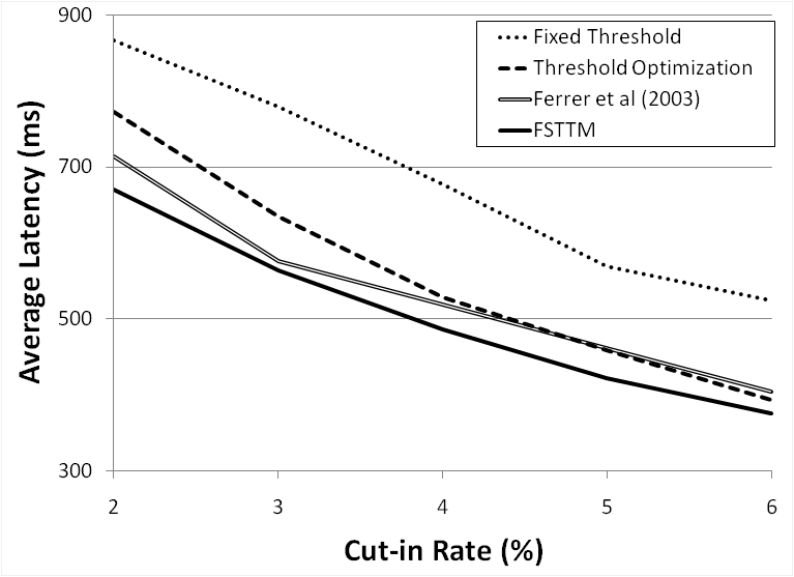


Figure 6.2: Cut-in / Latency Trade-off for Pause-based Endpointing in the FSTTM, compared with a fixed-threshold baseline, the threshold optimization approach described in Chapter 5, and the approach of Ferrer et al. [2003]. All confidence intervals for latency fall within $+/-4ms$. For the sake of clarity, we did not represent them on the graph.

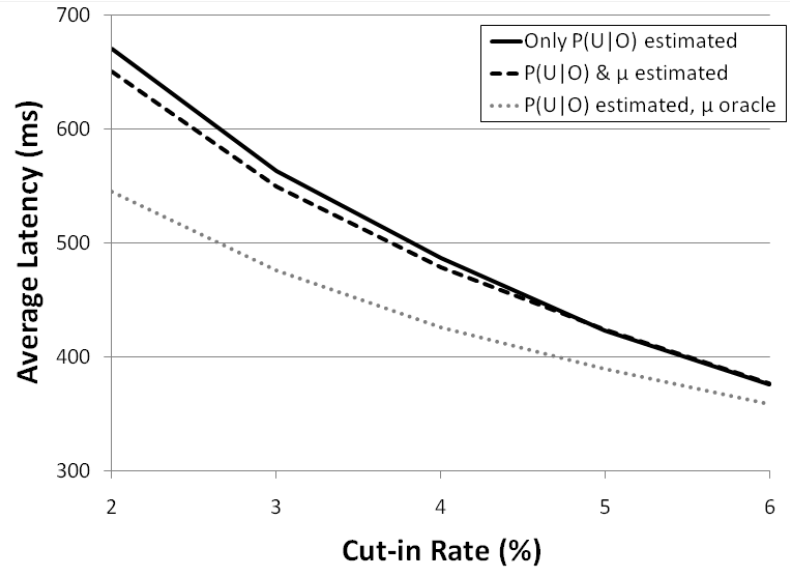


Figure 6.3: Cut-in / Latency Trade-off for Pause-based Endpointing in the FSTTM, compared with a fixed-threshold baseline and the threshold optimization approach described in Chapter 5. All confidence intervals for latency fall within $+/-4ms$. For the sake of clarity, we did not represent them on the graph.

endpointing decision algorithm, introduces delays (it takes at least 200 ms to detect a pause). However, contrarily to cluster-based threshold optimization, the FSTTM approach does not require that a pause be detected to be applied. Thus, we reformulate the problem as that of making a decision *every time a new partial recognition hypothesis is available* from the speech recognizer. Doing so allows the system to potentially endpoint even before a pause is detected. Note that our goal is not to generate overlapping speech but rather to better anticipate pauses. We redefine the endpoint of an utterance as the time at which the text of the current partial hypothesis is identical to what the final hypothesis would be, were we to wait for a significant pause to endpoint. The idea here is that if the text of the hypothesis does not change, the system has not gained any information by waiting, rather it lost time. Given this new goal, we define two cost matrices. The first one is used to make decisions before a pause is detected. The other is used once a pause longer than 200 ms has been detected by the speech recognizer, as that of Section 6.3. In both cases, we set the cost of grabbing the floor while the user still claims it at a constant C_U, as we did before. During a pause, the cost of waiting when the user has released the floor is again $C_G \cdot t$. The cost of waiting when no pause has been detected yet but the latest partial hypothesis is the same as the final one is set to another constant C_W. Thus the system grabs the floor when either there is not a pause longer than 200 ms at the end of the last partial hypothesis and

$$
\begin{aligned}
C(W,O) &\geq C(G,O) && (6.18)\\
P(F|O) \cdot C_W &\geq (1 - P(F|O)) \cdot C_U && (6.19)\\
P(F|O) \cdot C_W &\geq (1 - P(F|O)) \cdot C_U && (6.20)\\
P(F|O) &\geq \frac{C_U}{C_W + C_U} && (6.21)
\end{aligned}
$$

or there is pause of $t \geq 200ms$ at the end of the last partial hypothesis and, as before

$$
t \geq \frac{C_O}{C_G} \cdot P(t|O,U) \cdot \frac{1 - P(F|O)}{P(F|O)} \quad (6.22)
$$

As in Section 6.3, we need to estimate $P(F|O)$ at the beginning of pauses and $P(t|O,U)$, and, additionally, $P(F|O)$ at partial hypothesis during speech.

6.4.2 Estimating $P(F|O)$ During Speech

We use the same features and logistic regression model to learn $P(F|O)$ during speech, the goal being this time to predict whether a given partial hypothesis is the same as the final one in the training data. Because the log files of the corpus used in Section 6.3

Dialog state	Majority baseline	Classification error rate	Baseline log likelihood	Log likelihood
Open question	19.86%	16.51%	-0.4985	-0.3955
Closed question	32.26%	21.51%	-0.6288	-0.4947
Confirmation	35.95%	17.45%	-0.6531	-0.4003
Overall	38.99%	19.17%	-0.6687	-0.4495

Table 6.4: Performance of state-specific logistic regression for estimating $P(F|O)$ during speech segments.

lacked necessary information about partial hypotheses, we use a more recent corpus of 586 dialogs, collected between May, 4 and May, 14, 2008. The performance of the logistic regression for each dialog state and overall is shown in Table 6.4.

In contrast to the models learned at pauses, these models are much more accurate than the majority baseline. The best performance improvement is achieved in the confirmation state, where most utterances are predictable single words such as "YES" and "NO". The selected features, shown in table 6.5, overlap those selected for detection at pauses. One difference is the use of the duration of the final pause. While this model is only triggered during speech and not when a pause has been detected, our definition of "in speech" is that the final pause is shorter than 200 ms, so the model is able to extract information from durations between 0 and 190 ms, with longer pauses inducing a higher $P(F|O)$.

6.4.3 Batch Evaluation

We evaluated anytime endpointing using cross-validation on the May 2008 corpus. For this evaluation, we set the cost of not endpointing in speech when the floor is $FREE$, C_W to 1000. We then vary the cost of a cut-in C_U to compute the latency / cut-in rate trade-off curve for this model. We compare this approach to the fixed-threshold baseline, as well as the in-pause-only FSTTM approach (i.e. we set C_W to 0). Anytime FSTTM endpointing is consistently better than both baselines, improving over in-pause-only FSTTM endpointing by up to 17% for a cut-in rate of 5%. At that point the average latency of the Anytime-FSTTM endpointer is about 40% less than that of the fixed-threshold baseline.

To investigate the impact of in-speech endpointing on the overall performance of the model, we varied C_W while selecting C_U so as to keep a constant cut-in rate of 5%. The

Feature	Value range	Median	Open	Closed	Confirmation
Constant	-	-	0.3582	0.5852	0.1683
LM boundary score	$[-5.3, 2.4]$	-0.7	1.1947	1.1397	0.5888
Final pause duration	$[0, 190]$	28.7	0.0113	0.0107	0.0110
Avg nb words per turn	$[1, 27]$	1.9	-0.2656	-0.4181	-0.1368
Nb words in hyp	$[1, 35]$	1	-0.0557	0	-0.2555
Hyp has confirm	$0, 1$	0	0	0	2.6156
Hyp has disconfirm	$0, 1$	0	-1.1122	0.8555	2.0213
Hyp is non-und.	$0, 1$	0	-0.4535	-0.7820	-0.8857
Non-und. rate so far	$[0, 1]$	0.2	0	0.9434	0
User barge-in	$0, 1$	0	0	-0.1851	0

Table 6.5: Features selected by stepwise logistic regression to estimate $P(F|O)$ during speech segments and their coefficients (all non-null coefficients are non null with $p < 0.01$).

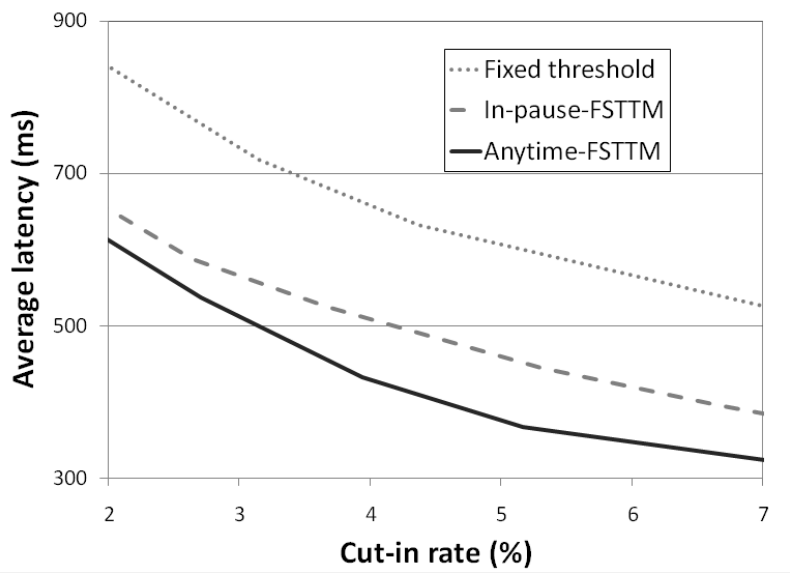


Figure 6.4: Cut-in / Latency Trade-off for Anytime Endpointing in the FSTTM, compared with a fixed-threshold baseline and the At-Pause endpointing. All confidence intervals for latency fall within $+/-4ms$. For the sake of clarity, we did not represent them on the graph.

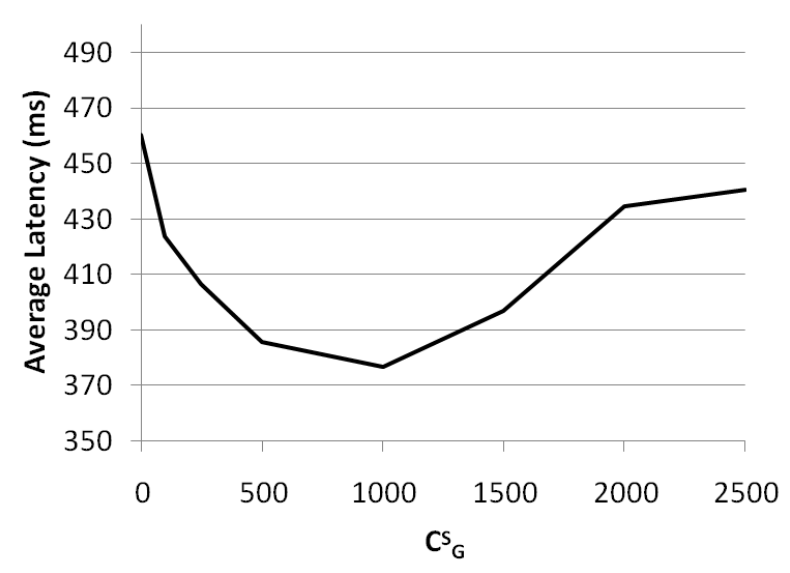


Figure 6.5: Average latency as a function of C_W for a fixed cut-in rate of 5%.

resulting curve, drawn on Figure 6.5, has a clear convex shape, with a minimum around $C_W = 1000$. With smaller values, the model endpoints less frequently in speech, resulting in "wasted" opportunities for faster response. On the other hand, for high C_W, the model aggressively endpoints in speech, leading to an increasing number of cut-ins for those utterances. To compensate and keep the overall cut-in rate at 5%, the in-pause decisions will be more conservative, leading to longer delays there. For other cut-in rates, we also found the optimal C_W value to lie between 500 and 1000. At optimal cut-in rates, 30 to 40% of the turns are endpointed during speech.

6.5 Interruption Detection

6.5.1 Problem Definition

Interruption detection is the problem of identifying when the user is trying to interrupt the system during a prompt (barge-in), and to rule out other cases. We call *false interruptions* cases where the system erroneously interrupts itself although the user was not trying to

barge in. In our FSTTM formalism, this problem translates into that of selecting between the Keep and the Release actions during a system prompt. We assume that at some point, the system knew the user was not claiming the floor at the same time that the system was. In other words, at some point in the past, the model state was $SYSTEM$ with probability 1. After that point, the uncertainty lies between the states $SYSTEM$ (user does not claim the floor) and $BOTH_S$ (user claims the floor, i.e. barges in). Potential user utterances are first detected by the VAD, which is equivalent to saying that $P(B_S|O) = 0$ when the VAD is in a non-speech state. In addition, we only make decisions when a new (non-empty) partial hypothesis is output by the ASR. Finally, we make the simplifying assumption that whenever the user addresses the system, they are claiming the floor. This treats backchannels as normal floor grabbing turns which is obviously wrong in the general case but, due to the constrained nature of most current systems, including Let's Go, is a reasonable first approximation.

Thus the problem is to distinguish user utterances directed at the system (state $BOTH_S$) from noises (i.e. VAD/ASR false positives) and other utterances (e.g. directed to another person), which correspond to state $SYSTEM$. Given the cost structure of table 6.1, interruptions are detected when:

$$\begin{aligned} C(K,O) &\geq C(R,O) && (6.23)\\ P(B_S|O) \cdot C_O &\geq P(S|O) \cdot C_S && (6.24)\\ P(B_S|O) \cdot C_O &\geq (1 - P(B_S|O)) \cdot C_S && (6.25)\\ P(B_S|O) &\geq \frac{C_S}{C_S + C_O} && (6.26) \end{aligned}$$

As for endpointing in speech, this amounts to a threshold on $P(B_S|O)$.

6.5.2 Estimating $P(B_S|O)$

Since we only make a decision, and thus compute $P(B_S|O)$, when a non-empty partial hypothesis is generated, we assume that all the acoustic evidence has already been taken into account by the VAD and the ASR. Therefore, at this stage, we focus on higher level cues. Specifically, our goal is to identify keywords which, when found in the partial hypothesis, provide evidence towards $SYSTEM$ or $BOTH_S$.

The intuition behind this approach is that barge-ins should exhibit specific words that are meaningful in the current dialog context, whereas false interruptions are likely to generate random words in the partial hypotheses. For example, during a prompt where the

	Match	Non-understanding
Barge-in	3891	1807
False interruption	378	1332

Table 6.6: Co-occurrence of Matches/Non-understandings and Manually Annotated Barge-ins/False interruptions.

system gives a bus time to the user, it is expected (and encouraged by help prompts) that the user would barge-in and produce utterances such as "WHEN IS THE NEXT BUS" or "WHEN IS THE PREVIOUS BUS". Therefore, if a partial hypothesis during that prompt contains "WHEN", it is probably evidence that it is a barge-in.

To identify these keywords, we used the Let's Go data from May, 3rd to September, 5th, 2008, which, overall, contained 158994 prompts, of which 28684 (18%) were interrupted (including both barge-ins and self interruptions), for a total of 234772 partial hypotheses. Each partial hypothesis was automatically marked with a binary label indicating if it belonged to an utterance that ultimately resulted in a match in the expectation agenda or a non-understanding. This is a very crude way of discriminating real interruptions from a false interruption (i.e. other speech or noise), since many genuine interruptions result in non-understandings, and some false interruptions result in a match. Indeed, we verified this by manually annotating a smaller corpus of 212 dialogs. Each input occurring during a system prompt that led to at least one non-empty partial hypothesis was marked as barge-in or false interruption. We excluded hypotheses that started 500 ms or less before the end of the prompt, since these tend to be genuine utterances but correspond to turn transitions with slight overlap rather than actual barge-in. For these cases, the system being about to finish speaking the prompt anyway, it is not necessary to make a barge-in decision. We thus have a corpus of 7408 partial hypotheses, 74% of which were marked as barge-in. On this data, we cross-tabulated non-understandings and barge-ins (see Table 6.6). We found that 91% of matches are barge-ins, whereas 78% of false interruptions are non-understandings. On the other hand, 58% of non-understandings are also barge-ins and 32% of barge-ins are non-understandings. These figures indicate that these automated match labels are by no means a good substitute for manual barge-in labels. However, for the purpose of identifying relevant keywords, we hoped that the large amount of data would compensate for this imperfection.

One property of these keywords is that they are likely to depend on the prompt, or at

least its dialog act, since both what the user says and the ASR language model depend on those. To take this into account, we segmented the data in 36 system dialogs acts such as "inform result" and "request departure_place". It is also likely that the keywords would depend on the duration of the partial hypothesis. In the example above, "BUS" might be a good keyword for long hypotheses (since both "WHEN IS THE NEXT BUS" and "WHEN IS THE PREVIOUS BUS" contain it), but not so for short ones since it is unlikely the user would actually start their utterance by "BUS". Therefore we further clustered each dialog act's partial hypotheses into 10 groups according to the estimated duration of speech in the hypothesis (based on time alignment information from the ASR). The first group contains hypothesis whose duration is between 0 and 100 ms, the second one between 100 and 200 and so on. The tenth group contains all hypotheses longer than 900 ms. As a result, we obtained 360 groups containing between 0 and 13927 partial hypotheses. 55 of these groups did not contain any partial hypothesis, while 165 contained 100 or more hypotheses. For each group, we computed the information gain of each word appearing in a partial hypothesis with respect to that hypothesis ultimately leading to a match. Information gain is defined as the difference between the entropy of a target variable and its conditional entropy on a conditioning variable. In our case, let us call M the binary variable that equals 1 when a given partial hypothesis belongs to what will ultimately be a matched utterance and 0 otherwise. Let W_i be another binary variable that equals 1 if keyword W_i appears in the hypothesis and 0 otherwise. The Information Gain of M given W_i is:

$$IG(M|W_i) = H(M) - H(M|W_i) \tag{6.27}$$

where the entropy of M $H(M)$ is

$$H(M) = P(M=0) \cdot log(P(M=0)) + P(M=1) \cdot log(P(M=1)) \tag{6.28}$$

and the conditional entropy $H(M|W_i)$ is defined as

$$H(M|W_i) = (P(W_i=0) \cdot H(M|W_i=0) + P(W_i=1) \cdot H(M|W-i=1) \tag{6.29}$$

with

$$H(M|W_i=x) = P(M=0|W_i=x) \cdot log(P(M=0|W_i=x) + P(M=1|W_i=x) \cdot log(P(M=1|W_i$$

We estimate the probabilities $P(M=y)$, $P(W_i=x)$ and $P(M=y|W_i=x)$ using the frequency counts in their 2x2 contingency table (smoothed by adding one to each cell). For each of the 360 groups, we rank the keywords by decreasing IG.

Tables 6.7 and 6.8 show the keywords for the two most commonly interrupted dialog acts. "explicit_confirm" indicates all prompts such as:

Hyp. duration	Nb of hyp	Non-und. rate	Top keywords
$0-100$	1437	12%	THE*, NO, TO*, OF*, YEAH
$100-200$	6196	21%	THE*, NO, YEAH, IT*, FROM*
$200-300$	11127	20%	YES, YEAH, THE*, NO, NEED*
$300-400$	13927	18%	YES, GOING*, NO, THE*, SHE*
$400-500$	11909	16%	YES, NO, GOING*, HELLO*, AM*
$500-600$	8373	14%	YES, NO, THE*, AM*, HELLO*
$600-700$	5398	19%	YES, AM*, NO, HELLO*, LEAVING*
$700-800$	3419	23%	YES, AM*, NO, HELLO*, TOLD*
$800-900$	2591	26%	HELLO*, AM*, WHAT*, P*, YES
$900-$	2300	26%	HELLO*, EN*, AM*, YES, ME*

Table 6.7: Barge-in keywords for the dialog act "explicit_confirm". * indicate words that signal self interruptions, all other words signal barge-ins.

Hyp. duration	Nb of hyp	Non-und. rate	Top keywords
$0-100$	328	61%	THE*, WH, BYE, NEW, WHEN
$100-200$	2476	68%	THE*, BYE, WHEN, WHAT, PRE
$200-300$	4534	54%	BUS*, WHEN, NEXT, THE*, PRE
$300-400$	3992	40%	NEXT, BUS*, IS, PRE, HELLO*
$400-500$	3607	36%	PREVIOUS, THE, NEXT, IS, OF*
$500-600$	3495	33%	NEXT, PREVIOUS, HELLO*, OF*, THE
$600-700$	3263	31%	NEXT, PREVIOUS, BUS, THE, HELLO*
$700-800$	3243	29%	NEXT, BUS, PREVIOUS, THE, DONE*
$800-900$	2787	26%	NEXT, BUS, PREVIOUS, THE, IS
$900-$	2918	26%	NEXT, BUS, PREVIOUS, THE, IS

Table 6.8: Barge-in keywords for the dialog act "request next_query". * indicate words that signal self interruptions, all other words signal barge-ins.

> Leaving from Downtown. Is this correct?

and

> The next bus. Did I get that right?

and "request next_query" corresponds to the following fixed prompt, after the system has given a bus time to the user:

> To get more information about buses related to this trip, you can say, when is the next bus, or, when is the previous bus. To ask about a different trip, you can say, start a new query. If you are finished, you can say goodbye.

Both dialog acts show an evolution over the duration of the hypothesis, albeit with different properties. In the "explicit_confirm" case, except for the very first time interval, the proportion of (future) non-understandings among partial hypotheses first decreases until reaching a minimum of 14% at the $500-600$ ms interval and finally increasing again. This shows that most genuine barge-ins have a duration of about 500 ms, which corresponds to the average duration of the words "YES" and "NO" alone. In contrast, self interruptions appear to be of two kinds. The first type is short noises, which explain the somewhat higher non-understanding rates between 100 ms and 400 ms. These noises are captured in partial hypotheses by the word "THE", which because of its short duration and fricative nature, tends to match non-speech noises. Another type of self interruption happen with hypotheses longer than 600 ms. It consists of user speech not directed to the system, background voices, or otherwise continuous noises, which explain the higher non-understanding rates at these durations. They are best captured by the word "HELLO", which because of its voiced nature tends to match background speech.

The non-understanding rates and keywords for the "request next_query" dialog act tell a different story. In this case, genuine barge-ins tend to be longer utterances like "WHEN IS THE NEXT BUS". Hence, shorter, non-speech events, along with genuine attempts at barging in that do not match the system's expectations (and hence will lead to a non-understanding) finish earlier than matches, which explains the decreasing ratio of future non-understandings. The "THE" keyword is also present here but with a dual pattern. At first, as in the explicit_confirm case, it indicates future non-understandings, while later on it becomes an indicator of matches because it appears in the frequent matches "WHEN IS THE NEXT BUS" and "WHEN IS THE PREVIOUS BUS". Also, the partial words "WH" and "PRE", who only appear in the ASR output because they were forgotten when we "cleaned up" the training corpus of the language model, are in fact good early predictors of matches since they announce the common valid words "WHEN" and "PREVIOUS".

The fact that words like "THE" and "HELLO", as well as partial words like "WH" and "PRE", are found to be informative indicates that these keywords not only capture what users normally say in actual interruptions but also the bias of the ASR towards certain words, in particular when confronted with noise. These specific keywords would of course be different when using a different speech recognizer or even just different acoustic and/or language models. The features would have to be recomputed for these new conditions.

For each partial hypothesis, we defined binary features indicating the presence of each of the top five keywords for each group. We also added one binary feature indicating whether the partial hypothesis itself is a match or a non-understanding. We trained 305 6-variable logistic regression models on the May-September 2008 data. The target variable is whether the partial hypothesis belongs to an utterance whose final hypothesis was a match given the current system expectations.

6.5.3 Batch Evaluation

We evaluated the models on the manually annotated corpus introduced in section 6.5.2. By varying the costs C_S and C_O we evaluate the average latency across a range of thresholds on $P(B_S|O)$. The results is shown on Figure 6.6. The most straightforward baseline, which is to interrupt the system prompt as soon as a non-empty partial hypothesis is generated leads to very fast decisions (233 ms on average) but a high false interruption rate (4.7% of all prompts are erroneously interrupted). This baseline (referred to as "first hypothesis") is not represented in Figure 6.6. A second baseline (referred to as "first match") uses semantic and discourse information by interrupting a system prompt at the first partial hypothesis that is a match in the current dialog context. This baseline, represented by a triangle on Figure 6.6, leads to slower reactions (543 ms on average). It is also less prone to false interruptions with a rate of 1.6%. Our FSTTM approach introduces a trade-off between latency and accuracy of the decision. At a false interruption rate of 1.6%, the average latency is 482 ms, which corresponds to a reduction of 61 ms or 11% over the first-match baseline.

6.6 Live Evaluation

To confirm the results of the batch evaluation, we implemented our FSTTM model in the Apollo Interaction Manager of the Olympus spoken dialog architecture (see Chapter 4). We let the public system's IM use either FSTTM as described above (condition **FSTTM**), or a fixed threshold for endpointing and interrupts system prompts as soon as a partial

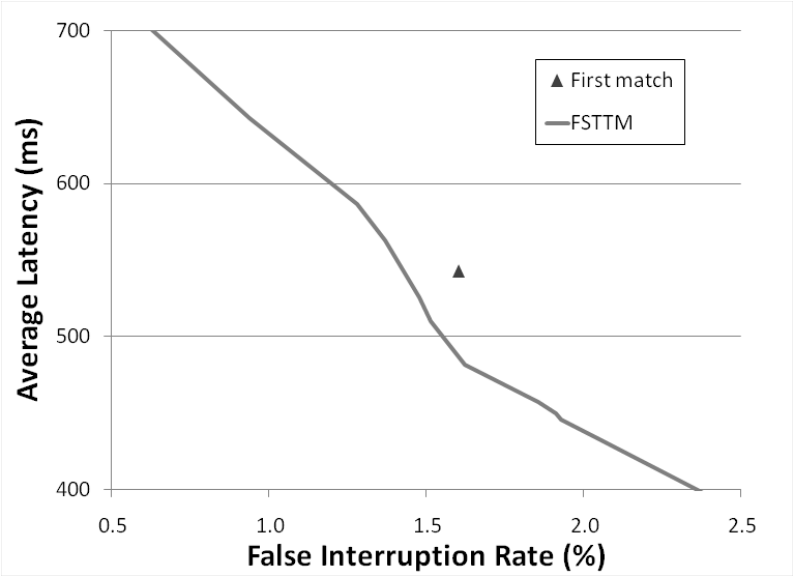
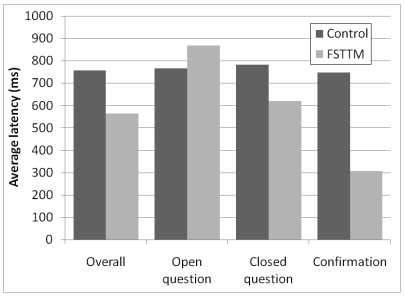


Figure 6.6: False Interruptions / Latency trade-off with the FSTTM and the first-match heuristic.

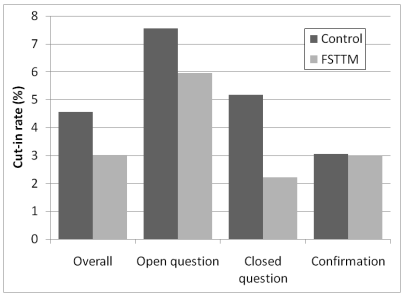
hypothesis matches the current expectation agenda (condition **control**). For FSTTM, we set the costs to the following values:

$$\begin{aligned} C_G &= 1 \\ C_U &= 5000 \\ C_W &= 500 \\ C_O &= 1 \\ C_S &= 10 \end{aligned}$$

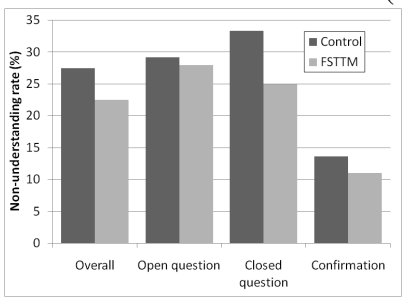
which, in the batch evaluation, corresponded to a cut-in rate of 6.3% and an average latency of 320 ms. For the control condition, we set the fixed endpointing threshold to 555 ms, which also corresponded to about 6.3% cut-ins. The barge-in costs correspond to setting a threshold on $P(U|O)$ at 0.9. This is a conservative decision that stems from our observations that the cost of interrupting certain prompts erroneously can have serious consequences on dialog success. This is the case, for example, when the user manages to specify their complete query to the system but simply does not hear the result because of an erroneous barge-in detection on the result prompt.



(a) Latency



(b) Cut-in rates



(c) Non-understanding rate

Figure 6.7: Live evaluation results. All the differences in latency are significant ($p < 1e-4$, using the Wilcoxon rank sum test). For cut-in rates, only the difference in the Confirmation state is significant ($p < 0.05$ using the binomial proportions test). For non-understanding rates, only the overall difference is statistically significant ($p < 0.05$).

Figure 6.7 shows average latency, cut-in rate and non-understanding rate for the control and experimental conditions. Our FSTTM approach improves over the baseline on all metrics, reducing average latency by 193 ms, cut-in rate by 1.5% and non-understanding rate by 4.9%. All the differences in latency are significant ($p < 1e-4$, using the Wilcoxon rank sum test). For cut-in rates, only the difference in the Confirmation state is significant ($p < 0.05$ using the binomial proportions test). For non-understanding rates, only the overall difference is statistically significant ($p < 0.05$). As already observed for the threshold optimization approach of Chapter 5, the latency reduction is largest for the confirmation dialog state (440 ms). On the other hands, both cut-in and non-understanding rates improve the most for the closed question state.

For interruption detection, the proportion of user utterances that start during a system prompt (barge-in rate) was not affected by the approach (33% in both cases). However the non-understanding rate for barge-ins was lower in the FSTTM condition (32%) than in the control condition (40%), but not significantly so.

In addition to these results, inspection of the live data revealed some issues with ASR-based pause detection. Indeed, a significant portion of the cut-ins are the result of poor pause detection, which happens when the PocketSphinx recognition engine fails to recognize silence or noise at the end of a partial utterance, artificially lengthening the last word instead.

6.7 Discussion

Both batch and live evaluation results confirm the effectiveness of our FSTTM approach in improving system responsiveness, both at the end of user turns and on user interruptions. For endpointing in particular, this approach improved significantly over the threshold optimization approach we proposed in Chapter 5. The language-model-based feature got the highest weight in the regression, indicating that in a domain such as Let's Go's, lexical cues are very informative for endpointing. The fact that the boundary LMs can be computed without any human transcription effort (since they are trained on the output of the recognizer) make them all the more appealing.

Lexical cues were also useful for interruption detection, albeit to a lesser extent. Our finding that the partial words "WH" and "PRE" are useful in early detection provides inspiration for future feature design. First, one could add more of these partial words to the language model, effectively leading to a mixed sub-word unit / word LM. This approach is likely to hurt recognition performance when scaled up, by introducing many highly confusable words in the ASR lexicon. Rather, one could imagine getting more fine-grain

information from the ASR. Currently, most ASR engines are able to produce word-level partial hypotheses corresponding to the most likely word sequence so far. It might be useful to also include the most likely next word(s) based on the word hypotheses currently active. This, combined with the problems encountered in ASR-based pause detection during the live evaluation, point to the need for more research on ASR engines not just as transcription tools producing sequences of words, but as dynamic components of interactive systems, providing a rich set of features to inform higher-level modules. Generally speaking, the performance of our model for interruption detection was not as good as for endpointing, particularly when compared with the first-match baseline. However, it should be noted that this is a rather strong baseline since it uses semantic and dialog expectation information. Ultimately, the live evaluation did show that the FSTTM approach did reduce non-understanding rate significantly. Another advantage of the FSTTM approach is that it allows tuning of the system's behavior by adjusting the costs, which heuristic approaches such as first-match do not allow.

Essentially, we showed that our FSTTM provides a simple, unified model of turn-taking that lends itself to data-driven optimization. While we described and evaluated specific cost structures and probability estimation techniques, the framework's flexibility opens it to other choices at many levels. In some ways, the FSTTM is an extension of previous classification based approaches to endpointing such as those proposed by Sato et al. [2002], Ferrer et al. [2003] and Takeuchi et al. [2004]. But it unifies these models with those proposed for backchannel / barge-in prediction and identification by, e.g. Ward and Tsukahara [2000] and Cathcart et al. [2003], as well as production by, e.g., Tsukahara [1998]. While much work is still needed before a comprehensive model of turn-taking for control of conversational agents can be designed, we believe the FSTTM provides a framework that can support research towards such a model.

Chapter 7

Conclusion

7.1 Summary of contributions

The research described in this dissertation addresses the problem of making spoken dialog systems more responsive to the user. Its main contributions are:

An architecture for semi-synchronous, event-driven, dialog management. While similar to other existing multi-layer spoken dialog systems architecture, the proposed architecture offer several advantages. First, it fully integrates a state-of-the-art dialog manager, with complex task representation, mixed-initiative, and grounding mechanisms, with a fine-grain low-level interaction module, the interaction manager. This integration is done in a fully task-independent fashion, allowing the use and evaluation of various turn-taking models in different tasks. Second, it centralizes information from all levels of dialog, including high-level information from the dialog manager, and low-level information from the ASR engine and other sensors, in a single structure, which allows turn-taking models to make use of a wide range of features to make their decisions. Third, it is freely available as open-source software, facilitating the implementation of full dialog systems and research on any aspect, including timing and turn-taking.

An algorithm to optimize endpointing thresholds. We designed a method to dynamically set the utterance endpointing threshold, using a model trained on past dialogs. Our evaluation shows that many features that are readily available in most spoken dialog systems can help improve responsiveness. In particular, we showed that, in information access systems, semantic information based on dialog context is the

most useful source of information for endpointing.

A decision theoretic model of turn-taking as a dynamic decision problem. Our model, the Finite-State Turn-Taking Machine, is, to our knowledge, the first general control mechanism for turn-taking in conversational systems implemented in a task-independent way. Offline and live evaluations showed its effectiveness. Beyond the specific cost structures and probabilistic models used in this work, the FSTTM is a new way of thinking about the turn-taking process, which formalizes it while keeping a high degree of flexibility. While this work has focused on dyadic, telephone, information access dialogs, the FSTTM can be applied to many other types of conversations.

A demonstration of the validity of our approach on real deployed systems All of the experiments in this work were conducted on the Let's Go system, a publicly available system that has provided bus information to Pittsburghers through more than 60000 telephone calls so far. Not only did Let's Go provide large amounts of data to train models, but it also allowed us to test our approaches directly on hundreds of dialogs with genuine user goals. In addition, all the data collected with the Let's Go system is freely available for other researchers. While this research paradigm can be further refined, in particular by combining controlled studies with live experiments, this is, to our knowledge, the first work that exploits an academic deployed system to this extent.

7.2 Possible extensions

Throughout this thesis, we have designed, implemented, and evaluated methods to make spoken dialog systems more responsive. While the results were positive, we believe that there is still a long way until the interaction with artificial conversational agents is as efficient and comfortable as human-human conversation. This work could be taken towards a number of directions.

First, while we designed and implemented the Olympus 2 architecture to be truly task-independent and open to many modalities, we have only tested it in the case of dyadic telephone conversations in this thesis. Application of Olympus 2 to other domains involving multiple participants and multiple modalities is underway in the TeamTalk [Harris et al., 2005]. That project and others in the future offer the opportunity to explore turn-taking and timing issues in much more complex settings than the Let's Go system does. There is no doubt that the current version of the architecture will need to be modified to truly

accommodate a wide range of domains and interaction modalities. We hope that this will ultimately result in a generic framework for turn-taking applied and tested on complete, working, and if possible, deployed, dialog systems.

Second, as already explained, the Finite-State Turn-Taking Machine lends itself to many applications and improvements. While we hope that the general ideas of finite-state floor modeling and decision-theoretic action selection will stand the test of time, its inner workings and computational details are open to change. For example, new features that are most useful for turn-taking, in particular using prosody and visual cues, can be explored and incorporated. The probability models that derive state probabilities from these features can be modified from the current fairly direct logistic regression-based models.

One very active research area in the field of spoken dialog systems are statistical models for dialog management [Singh et al., 2002, Henderson et al., 2005, Williams and Young, 2007]. These models typically make use of reinforcement learning to train a policy that maps system actions to dialog states. A very similar approach could be applied to the FSTTM, which would provide for a principled way to track of the uncertainty over the floor state over time, as well as a framework to integrate more context in the turn-taking decisions. For example, a reinforcement learning algorithm could learn a policy that triggers a different action after several floor conflicts (when the user and system repeated start speaking at the same time and interrupt themselves).

In general, the issue of turn-taking error handling is a crucial component of a really responsive system. With good floor conflict solving abilities, a system could afford to take more aggressively turn-taking decisions, knowing that it could recover at low cost if the decision was wrong. It is indeed very common, and not an issue, for people to start speaking at the same time and resolving the conflict in a matter of milliseconds.

Another aspect that can lend itself to improvement is cost structure. Our design of the cost matrix was motivated by intuition and the will to keep it as simple as possible. The result we have obtained indicate that our choices were reasonable. On the other hand, one can imagine more complex cost structures which, instead of optimizing for objective average latency as we did, are directly optimizing the naturalness of the interaction. To this end, perceptual and/or user studies could be crucial in order to determine the actual cost of latency and turn-taking errors (cut-ins and false interruptions) for users. These costs might depend on time, but also on other contextual information such as dialog state, user and system utterance semantic content, and user personality. The flexibility of the FSTTM approach, which makes it amenable to such perception-driven costs is one of its most significant strengths.

Applying the FSTTM to multimodal systems will also provide interesting challenges

and opportunities. How can one use visual and other non-speech cues to optimize turn-taking? How do the different modalities interplay in terms of floor? For example, when can the visual modality be used to convey information when the "acoustic" floor is not available? These are all questions that we will need to answer as systems get more sophisticated and more embedded in the real world.

Multi-party conversation is another case where turn-taking becomes both more complex and more crucial to the successful unfolding of a dialog. What is the state structure in a multi-party conversation? Can the floor be modeled in a way that is independent of the number of participants? The focus of our work is the control of an agent rather than the analysis of naturally occurring human-human dialogs. Yet, multi-party conversations give rise to situations where a given system is simply listening to two other parties talking with each other. How can we model these cases? If the two participants in question are human, how much understanding is necessary to track the floor state? How does the state relate to social relationships and obligations in such context?

The problems defining the floor state, dynamically tracking it and acting in the most efficient and/or natural way are, at this point, open research questions in the general case. We hope that the FSTTM can provide a framework in which these can be investigated in the context of artificial conversational agents.

Bibliography

H. Ai, D. Litman, K. Forbes-Riley, M. Rotaru, J. Tetreault, and A. Purandare. Using system and user performance features to improve emotion detection in spoken tutoring dialogs. In *Proc. Interspeech 2006*, Pittsburgh, PA, USA, 2006. 5.2

H. Ai, A. Raux, D. Bohus, M. Eskenazi, and D. Litman. Comparing spoken dialog corpora collected with recruited subjects versus real users. In *8th SIGDial Workshop on Discourse and Dialog*, Antwerp, Belgium, 2007. 3.2

G. S. Aist. Expanding a time-sensitive conversational architecture for turn-taking to handle content-driven interruption. In *Proc. ICSLP 1998*, Sydney, Australia, 1998. 2.2.4

G. S. Aist and J. Mostow. Adapting human tutorial interventions for a reading tutor that listens: Using continuous speech recognition in interactive educational multimedia. In *Proc. CALL '97 Conference on Multimedia*, Exeter, England, 1997. 2.2.4

J. F. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. Towards a generic dialogue shell. *Natural Language Engineering*, 6(3):1–16, 2000. 2.2.3

J. F. Allen, G. Ferguson, and A. Stent. An architecture for more realistic conversational systems. In *Proc. Intelligent User Interfaces 2001 (IUI-2001)*, pages 1–8, Santa Fe, NM, 2001. 2.2.3

J. F. Allen, G. Ferguson, A. Stent, S. Stoness, M. Swift, L. Galescu, N. Chambers, E. Campana, and G. S. Aist. Two diverse systems built using generic components for spoken dialogue (recent progress on trips). In *Interactive Demonstration Track, Association of Computational Linguistics Annual Meeting*, Ann Arbor, MI, 2005. 2.2.3

J. F. Allen and C. R. Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15:143–178, 1980. 1.2, 1.3.4

G. T. M. Altmann and M. J. Steedman. Interaction with context during human sentence processing. *Cognition*, 30(3):191–238, 1988. 2.2.1

J. Ang, R. Dhillon, A. Krupski, E. Shriberg, and A. Stolcke. Prosody-based automatic detection of annoyance and frustration in human-computer dialog. In *Proc. ICSLP 2002*, Denver, CO, USA, 2002. 5.2

ACL Workshop on Incremental Parsing, 2004. *Proc. ACL Workhop on Incremental Parsing: Bringing Engineering and Cognition Together*, Barcelona, Spain, 2004. Association for Computational Linguistics. 1.3.3, 2.2.1

M. Bacchiani, F. Beaufays, J. Schalkwyk, M. Schuster, and B. Strope. Deploying goog-411: Early lessons in data, measurement, and testing. In *ICASSP-08*, Las Vegas, NE, USA, 2008. 3.2

G. W. Beattie. Turn-taking and interruption in political interviews: Margaret Thatcher and Jim Callaghan compared and contrasted. *Semiotica*, 39(1-2):93–114, 1982. 5.3.4

G. W. Beattie. *Talk: An Analysis of Speech and Non-Verbal Behaviour in Conversation*. Open University Press, 1983. 2.1.2

A. Black. Clustergen: A statistical parametric synthesizer using trajectory modeling. In *Interspeech*, Pittsburgh, PA, USA., 2006. 1.3.6

D. Bohus. *Error Awareness and Recovery in Conversational Spoken Language Interfaces*. PhD thesis, Carnegie Mellon University, 2007. 4.3.4

D. Bohus, A. Raux, T. Harris, M. Eskenazi, and A. Rudnicky. Olympus: an open-source framework for conversational spoken language interface research. In *HLT-NAACL 2007 workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology*, Rochester, NY, USA, 2007. 3.3, 4.4

D. Bohus and A. Rudnicky. Integrating multiple knowledge sources for utterance-level confidence annotation in the CMU Communicator spoken dialog system. Technical Report CS-190, Carnegie Mellon University, Pittsburgh, PA, USA, 2002. 3.3.1, 5.2, 5.3.4, 5.3.4

D. Bohus and A. Rudnicky. RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda. In *Eurospeech03*, Geneva, Switzerland, 2003. (document), 1.3.4, 3.3.2, 4.3.4, 4.3, 4.5, 5.3.4, 6.3.2

D. Bohus and A. Rudnicky. Error handling in the ravenclaw dialog management architecture. In *Proc. HLT/EMNLP 2005*, Vancouver, BC, 2005. 1.1

R. Peter Bonasso, David Kortenkamp, David P. Miller, and Marc Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:237–256, 1997. 2.2.1

P. T. Brady. A model for generating on-off speech patterns in two-way conversation. *The Bell System Technical Journal*, 48:2445–2472, 1969. (document), 6.2.1, 6.1, 6.2.1, 6.2.2

R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1985. 2.2.1

M. Bull. *The Timing and Coordination of Turn-Taking*. PhD thesis, University of Edinburgh, 1997. 3.5.4

M. Bull and M. Aylett. An analysis of the timing of turn-taking in a corpus of goal-oriented dialogue. In *ISCLP 98*, pages 1175–1178, Sydney, Australia, 1998. 3.5.4, 5.3.4

J. Carletta, S. Isard, G. Doherty-Sneddon, A. Isard, J. C. Kowtko, and A. H. Anderson. The reliability of a dialogue structure coding scheme. *Computational Linguistics*, 23 (1):13–31, March 1997. 3.5.4

J. Cassell, T. Bickmore, L. Campbell, H. Vilhjalmsson, and H. Yan. More than just a pretty face: conversational protocols and the affordances of embodiment. *Knowledge-Based Systems*, 14:55–64, 2001. 6.2.1, 6.2.2

N. Cathcart, J. Carletta, and E. Klein. A shallow model of backchannel continuers in spoken dialogue. In *Proc. 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL10)*, pages 51–58, Budapest, Hungary, 2003. 2.1.3, 6.7

W. L. Chafe. *Talking Data: Transcription and Coding Methods for Language Research*, chapter Prosodic and Functional Units of Language, pages 33–43. Lawrence Erlbaum, 1992. 2.1.2, 5.3.4

Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18: 33–44, 1997. 1.3.3

P. M. Clancy, S. A. Thompson, R. Suzuki, and H. Tao. The conversational use of reactive tokens in english, japanese, and mandarin. *Journal of Pragmatics*, 26:355–387, 1996. 2.1.3, 5.3.4

H.H. Clark. *Using language*. Cambridge University Press, 1996. 1.2, 6.2.3

H.H. Clark and E.F. Schaefer. Contributing to discourse. *Cognitive Science*, 13:259–294, 1989. 1.2

P. Clarkson and R. Rosenfeld. Statistical language modeling using the cmu-cambridge toolkit. In *Eurospeech97*, Rhodes, Greece, 1997. 6.3.2

P.R. Cohen and C.R. Perrault. elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212, 1979. 1.2

D. Cournapeau, T. Kawahara, K. Mase, and T. Toriyama. Voice activity detector based on enhanced cumulant of lpc residual and on-line em algorithm. In *Interspeech 2006*, Pittsburgh, USA, 2006. 1.3.1

G. Damnati, F. Béchet, and R. De Mori. Spoken language understanding strategies on the france telecom 3000 voice agency corpus. In *ICASSP 2007*, 2007. 3.2

C. Darves and S. Oviatt. Adaptation of users' spoken dialogue patterns in a conversational interface. In *ICSLP 2002*, 2002. 3.5.5

R. Denny. *Interaction Structure and Strategy*, chapter Pragmatically Marked and Unmarked Forms of Speaking-Turn Exchange, pages 135–174. Cambridge University Press, 1985. 2.1.3

K. Dohsaka and A. Shimazu. System architecture for spoken utterance production in collaborative dialogue. In *Working Notes of IJCAI 1997 Workshop on Collaboration, Cooperation and Conflict in Spoken Dialogue Systems*, Nagoya, Japan, 1997. 2.2.3

S. Duncan. Some signals and rules for taking speaking turns in conversations. *Journal of Personality and Social Psychology*, 23(2):283–292, 1972. 2.1.2, 5.3.4

S. Duncan and D. W. Fiske. *Interaction Structure and Strategy*, chapter The Turn System, pages 43–64. Cambridge University Press, 1985. 2.1.3

T. Dutoit, V. Pagel, N. Pierret, O. van der Vreken, and F. Bataille. The MBROLA project: Towards a set of high-quality speech synthesizers free of use for non-commercial purposes. In *ICSLP96*, volume 3, pages 1393–1397, Philadelphia, PA., 1996. http://tcts.fpms.ac.be/synthesis/mbrola.html. 1.3.6

Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, 1970. ISSN 0001-0782. 1.3.3

J. Edlund, G. Skantze, and R. Carlson. Higgins - a spoken dialogue system for investigating error handling techniques. In *Proc. ICSLP*, Jeju, Korea, 2004. 1.1

F. Farfán, H. Cuayáhuitl, and A. Portilla. Evaluating dialogue strategies in a spoken dialogue system for email. In *Proc. IASTED Artificial Intelligence and Applications*, Manalmádena, Spain, 2003. 1.1

G. Ferguson and J. F. Allen. Trips: An integrated intelligent problem-solving assistant. In *Proc. Fifteenth National Conference on AI (AAAI-98)*, pages 26–30, 1998. 2.2.3

L. Ferrer, E. Shriberg, and A. Stolcke. A prosody-based approach to end-of-utterance detection that does not require speech recognition. In *ICASSP*, Hong Kong, 2003. (document), 5.1, 5.2, 5.3.4, 5.3.4, 5.5.2, 5.6, 6.3.4, 6.2, 6.7

K. Forbes-Riley, M. Rotaru, D. J. Litman, and J. Tetreault. Exploring affect-context dependencies for adaptive system development. In *Proc. HLT/NAACL 2007*, 2007. 5.3.4

C. E. Ford and S. A. Thompson. *Interaction and Grammar*, chapter Interactional Units in Conversation: Syntactic, Intonational, and Pragmatic Resources for the Management of Turns, pages 134–184. Cambridge University Press, 1996. 2.1.2, 2.1.2, 5.3.4

H. Furo. *Turn-Taking in English and Japanese. Projectability in Grammar, Intonation, and Semantics.* Routeledge, 2001. 2.1.2, 2.1.2, 2.1.3, 5.3.4, 5.3.4, 5.3.4

M. Gabsdil and O. Lemon. Combining acoustic and pragmatic features to predict recognition performance in spoken dialogue systems. In *Proc. ACL 2004*, Barcelona, Spain, 2004. 5.2

C. Goodwin. *Conversational Organization: Interaction between Speakers and Hearers*. Academic Press, 1981. 2.1.2

B. J. Grosz and C. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986. 1.2, 1.3.4

D. Guillevic, S. Gandrabur, and Y. Normandin. Robust semantic confidence scoring. In *Proc. ICSLP 2002*, Denver, CO, USA, 2002. 5.2

D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987. 2.2.4, 6.2.1

T. K. Harris, S. Banerjee, and A. Rudnicky. Heterogeneous multi-robot dialogues for search tasks. In *Proc. AAAI Workshop on Dialogical Robots: Verbal Interaction with Embodied Agents and Situated Devices*, Stanford, CA, 2005. 4.5, 7.2

H. Hassan, A. Crespo, and J. Simó. Flexible real-time architecture for hybrid mobile robotic applications. In *Proc. 9th International Federation of Automatic Control Symposium*, Budapest, Hungary, 2000. 2.2.1

J. Henderson, O. Lemon, and K. Georgila. Hybrid reinforcement/supervised learning for dialogue policies from communicator data. In *IJCAI Workshop*, 2005. 7.2

R. Higashinaka, K. Sudoh, and M. Nakano. Incorporating discourse features into confidence scoring of intention recognition results in spoken dialogue systems. In *Proc. ICASSP 2005*, 2005. 5.2, 5.3.4

J. Hirschberg, D. Litman, and M. Swerts. Prosodic and other cues to speech recognition failures. *Speech Communication*, 2004. 5.2

X. Huang, F. Alleva, H.-W. Hon, K.-F. Hwang, M.-Y. Lee, and R. Rosenfeld. The SPHINX-II speech recognition system: an overview. *Computer Speech and Language*, 7(2):137–148, 1992. 3.3.1

X.D. Huang, A. Acero, and H.W. Hon. *Spoken Language Processing*. Prentice Hall PTR, 2001. 1.3.2

D. Huggins-Daines, M. Kumar, A. Chan, A. W Black, M. Ravishankar, and A. I. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *ICASSP 2006*, Toulouse, France., 2006. 3.3.1

J. Hulstijn and G.A.W. Vreeswijk. Turntaking: a case for agent-based programming. Technical Report UU-CS-2003-045, Institute of Information and Computing Sciences, Utrecht University, Utrecht, NL, 2003. 2.2.4

A. Hunt and A. Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *ICASSP96*, volume 1, pages 373–376, Atlanta, Georgia, 1996. 1.3.6

J. Jaffe and S. Feldstein. *Rhythms of Dialogue*. Academic Press, 1970. (document), 1.2, 3.5.4, 5.3.4, 6.2.1, 6.1, 6.2.1, 6.2.2

Y. Kamide, G. T. M. Altmann, and S. L. Haywood. The time-course of prediction in incremental sentence processing: Evidence from anticipatory eye movements. *Journal of Memory and Language*, 49:133–156, 2003. 2.2.1

Y. Kato, S. Matsubara, Toyama K., and Y. Inagaki. Incremental dependency parsing based on headed context free grammar. *Systems and Computers in Japan*, 36(2):84–97, 2005. 1.3.3, 2.2.1

A. Kendon. Some functions of gaze direction in social interaction. In *Acta Psychologica*, volume 26, 1967. 2.1.2

J. Ko, F. Murase, T. Mitamura, E. Nyberg, T. Tateishi, and I. Akahori. Cammia - a context-aware spoken dialog system. In *ASRU 2005*, San Juan, Puerto Rico., 2005. 1.4.3

H. Koiso, Y. Horiuchi, S. Tutiya, A. Ichikawa, and Y. Den. An analysis of turn-taking and backchannels based on prosodic and syntactic features in japanese map task dialogs. *Language and Speech*, 41(3-4):295–321, 1998. 2.1.2, 2.1.3, 5.3.4

J. Kominek and A. Black. The CMU ARCTIC speech databases for speech synthesis research. Technical Report CMU-LTI-03-177 http://festvox.org/cmu_arctic/, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, 2003. 3.3.3

F. Kronild. Turn taking for artificial conversational agents. In *Cooperative Information Agents X*, Edinburgh, UK, 2006. 2.2.4, 6.2.1, 6.2.2

F. Kronlid. *Steps towards Multi-Party Dialogue Management*. PhD thesis, University of Gothenburg, 2008. 2.2.4

S. Larsson and D. Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering*, 6:323–340, 2000. 1.3.4

Alon Lavie and Masaru Tomita. Glr* - an efficient noise-skipping parsing algorithm for context-free grammars. In *In Proceedings of the Third International Workshop on Parsing Technologies*, pages 123–134, 1993. 1.3.3

O. Lemon, L. Cavedon, and B. Kelly. Managing dialogue interaction: A multi-layered approach. In *Proc. SIGdial Workshop 2003*, Sapporo, Japan, 2003. 2.2.4, 4.5

M. Lennes and H. Anttila. Prosodic features associated with the distribution of turns in finnish informal dialogues. In Petri Korhonen, editor, *The Phonetics Symposium 2002*, volume Report 67, pages 149–158. Laboratory of Acoustics and Audio Signal Processing, Helsinki University of Technology, 2002. 5.3.4

W. J. M. Levelt. *Speaking: from Intention to Articulation*. MIT Press, 1993. 2.1.2

J. Liscombe, G. Riccardi, and D. Hakkani-Tr. Using context to improve emotion detection in spoken dialog systems. In *Proc. Interspeech 2005*, Lisbon, Portugal, 2005. 5.2, 5.3.4

M. Marzinzik and B. Kollmeier. Speech pause detection for noise spectrum estimation by tracking power envelope dynamics. *IEEE Trans. on Speech and Audio Processing*, 10: 109–118, 2002. 1.3.1

P. McCullagh and J. Nelder. *Generalized Linear Models*. Chapman and Hall, London, 1989. 6.3.3

D. Mori, S. Matsubara, and Y. Inagaki. Incremental parsing for interactive natural language interface. In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Tucson, AZ, 2001. 1.3.3, 2.2.1

J. P. Muller. *The Design of Intelligent Agents: A Layered Approach*. Springer, 1996. 2.2.1

M. Nakano, K. Dohsaka, N. Miyazaki, J. Hirasawa, M. Tamoto, M. Kawamori, A. Sugiyama, and T. Kawabata. Handling rich turn-taking in spoken dialogue systems. In *Eurospeech*, Budapest, Hungary, 1999a. 2.2.3

M. Nakano, N. Miyazaki, J. Hirasawa, K. Dohsaka, and T. Kawabata. Understanding unsegmented user utterances in real-time spoken dialogue systems. In *ACL*, College Park, MD, 1999b. 2.2.1

M. Nakano, N. Miyazaki, N. Yasuda, N. Sugiyama, J. Hirasawa, K. Dohsaka, and K. Aikawa. Wit: A toolkit for building robust and real-time spoken dialogue systems. In *1st SIGdial Workshop*, Hong-Kong, 2000. 2.2.3

E. Nemer, R. Goubran, and S. Mahmoud. Robust voice activity detection using higher-order statistics in the LPC residual domain. *IEEE Trans. on Speech Audio Processing*, 9:217–231, 2001. 1.3.1

A. Oh and A. Rudnicky. Stochastic language generation for spoken dialogue systems. In *ANLP/NAACL 2000 Workshop on Coversational Systems*, pages 27–32, Seattle, WA, 2000. 1.3.5

B. Oreström. *Turn-Taking in English Conversation*. CWK Gleerup, Lund, 1983. 2.1.2, 5.3.4, 5.3.4

R. Porzel and M. Baudis. The tao of chi: Towards effective human-computer interaction. In *HLT/NAACL 2004*, Boston, MA, 2004. 1.1

J. Ramírez, J. M. Górriz, and J. C. Segura. *Robust Speech Recognition and Understanding*, chapter Voice Activity Detection. Fundamentals and Speech Recognition System Robustness, pages 1–22. New York: Academic, 2007. ISBN 978-3-902613-08-0. 1.3.1

A. Raux, D. Bohus, B. Langner, A. W. Black, and M. Eskenazi. Doing research on a deployed spoken dialogue system: One year of Let's Go! experience. In *Proc. Interspeech 2006*, Pittsburgh, PA, USA, 2006. 1.4.1, 3.2, 3.3, 3.3.2

A. Raux, B. Langner, A. Black, and M. Eskenazi. LET'S GO: Improving spoken dialog systems for the elderly and non-native. In *Eurospeech03*, Geneva, Switzerland, 2003. 1.4.1, 3.3

A. Raux, B. Langner, D. Bohus, A. W. Black, and M. Eskenazi. Let's Go Public! taking a spoken dialog system to the real world. In *Proc. Interspeech 2005*, Lisbon, Portugal, 2005. 1.4.1, 3.3

C. Rich, N.B. Lesh, J. Rickel, and A. Garland. Architecture for generating collaborative agent responses,. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2002. 1.3.4, 4.5

C. P. Rose, A. Roque, D. Bhembe, and K. VanLehn. An efficient incremental architecture for robust interpretation. In *Proc. Human Languages Technology Conference*, San Diego, CA, 2002. 1.3.3, 2.2.1

M. Rotaru and D. J. Litman. Discourse structure and speech recognition problems. In *Proc. Interspeech 2006*, Pittsburgh, PA, USA, 2006a. 5.3.4

M. Rotaru and D. J. Litman. Exploiting discourse structure for spoken dialogue performance analysis. In *Proc. EMNLP 2006*, Sydney, Australia, 2006b. 5.3.4

H. Sacks, E. A. Schegloff, and G. Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4):696–735, 1974. 1.2, 1.5, 2.1.1, 2.1.2, 2.2.4, 5.2, 6.2.1, 6.2.3

R. Sato, R. Higashinaka, M. Tamoto, M. Nakano, and K. Aikawa. Learning decision trees to determine turn-taking by spoken dialogue systems. In *ICSLP 2002*, Denver, CO, 2002. 2.2.3, 5.2, 6.7

D. Schaffer. The role of intonation as a cue to turn taking in conversation. *Journal of Phonetics*, 11:243–257, 1983. 2.1.2

S. Singh, D. Litman, M. Kearns, and M. Walker. Optimizing dialogue management with reinforcement leaning: experiments with the njfun system. *Journal of Artificial Intelligence*, 16:105–133, 2002. 7.2

Kare Sjolander. The snack sound toolkit. http://www.speech.kth.se/snack/, 2004. 5.3.4

M.-L. Sorjonen. *Interaction and Grammar*, chapter On repeats and responses in Finnish conversations, pages 277–327. Cambridge University Press, 1996. 2.1.2

S. C. Stoness, J. Tetreault, and J. Allen. Incremental parsing with reference interaction. In *Proc. ACL Workshop on Incremental Parsing*, pages 18–25, Barcelona, Spain, 2004. 2.2.1

M. Takeuchi, N. Kitaoka, and S. Nakagawa. Timing detection for realtime dialog systems using prosodic and linguistic information. In *Proc. Speech Prosody 04*, Nara, Japan, 2004. 5.2, 6.7

K. R. Thorisson. *Communicative Humanoids: A Computational Model of Psychosocial Dialogue Skills*. PhD thesis, Massachusetts Institute of Technology, 1996. 2.2.2, 4.5

K. R. Thorisson. A mind model for communicative humanoids. *International Journal of Applied Artificial Intelligence*, 13(4-5):449–486, 1999. 2.2.2

K. R. Thorisson. *Multimodality in Language and Speech Systems*, chapter Natural Turn-Taking Needs No Manual: Computational Theory and Model, From Perception to Action, pages 173–207. Kluwer Academic Publishers, 2002. 2.2.2, 4.5, 6.2.1, 6.2.2

K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *ICASSP*, volume 3, pages 1315–1318, 2000. 1.3.6

D. R. Traum and J. F. Allen. Discourse obligations in dialogue. In *Proc. ACL-94*, pages 1–8, 1994. 1.2

Wataru Tsukahara. An algorithm for choosing japanese acknowledgments using prosodic cues and context. In *In International Conference on Spoken Language Processing*, pages 691–694, 1998. 6.7

L. K. Tyler and W. D. Marlsen-Wilson. The on-line effects of semantic context on syntactic processing. *Journal of Verbal Learning and Verbal Behaviour*, 16:683–692, 1977. 2.2.1

K. Wang. Semantic synchronous understanding for robust spoken language understanding applications. In *Proc. ASRU 2003*, St Thomas, US Virgin Islands, 2003. 2.2.1

N. Ward, A. Rivera, K. Ward, and D. Novick. Root causes of lost time and user stress in a simple dialog system. In *Interspeech 2005*, Lisbon, Portugal, 2005. 1.1, 1.4.1

N. Ward and W.. Tsukahara. Prosodic features which cue back-channel responses in english and japanese. *Journal of Pragmatics*, 32:1177–1207, 2000. 2.1.3, 6.7

W. Ward. Understanding spontaneous speech: the phoenix system. In *ICASSP 91*, pages 365–367, 1991. 1.3.3

F. 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. A conversational dialogue system for cognitively overloaded users. In *ICSLP 2004*, Jeju Island, Korean, 2004. 1.4.3

W. Wesseling and R.J.J.H. van Son. Timing of experimentally elicited minimal responses as quantitative evidence for the use of intonation in projecting TRPs. In *Interspeech 2005*, pages 3389–3392, Lisbon, Portugal, 2005. 2.1.2

Graham Wilcock and Kristiina Jokinen. Generation models for spoken dialogues. In *In Natural Language Generation in Spoken and Written Dialogue, Papers from the 2003 AAAI Spring Symposium*, pages 159–165, 2003. 1.3.5

J. Williams and S. Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):393–422, 2007. 7.2

M. Wiren. *Studies in Incremental Natural Langauge Analysis*. PhD thesis, Linkoping University, 1992. 1.3.3, 2.2.1