A MULTI-LAYER ARCHITECTURE FOR SEMI-SYNCHRONOUS EVENT-DRIVEN DIALOGUE MANAGEMENT

Antoine Raux and Maxine Eskenazi

Language Technologies Institute
School of Computer Science
Carnegie Mellon University

ABSTRACT

We present a new architecture for spoken dialogue systems that explicitly separates the discrete, abstract representation used in the high-level dialogue manager and the continuous, real-time nature of real world events. We propose to use the concept of conversational floor as a means to synchronize the internal state of the dialogue manager with the real world. To act as the interface between these two layers, we introduce a new component, called the Interaction Manager. The proposed architecture was implemented as a new version of the Olympus framework, which can be used across different domains and modalities. We confirmed the practicality of the approach by porting Let's Go, an existing deployed dialogue system to the new architecture.

Index Terms— spoken dialogue systems, multi-layer architectures, conversational interfaces

1. INTRODUCTION

After several decades of research and development effort in the realm of practical spoken dialogue systems, the technologies have matured enough to allow widespread use of such systems. Still, the approaches that have permitted the creation of working systems have left many issues unsolved. As a result, spoken conversation with artificial agents often remains unsatisfactory, seldom natural. Perhaps the most prominent issue is the quality of automatic speech recognition (ASR), which often results in misunderstandings that, in turn, lead to dialogue breakdowns. There are several approaches to overcome this problem: improving recognition technology [1], limiting the interaction in some way [2], or endowing systems with error handling capabilities to smoothly recover from misrecognitions [3, 4]. While the first approach, improving ASR, seems the most obvious direction, actual improvements have been incremental. In addition, even as recognition accuracy reaches acceptable levels on simple tasks, applications target more and more complex domains, requiring ever higher performance from ASR engines. Robustness in a wide variety of conditions (e.g. with noisy or highly conversational speech) remains a significant challenge [5]. Short of perfect speech recognition, the other two approaches provide ways to cope with recognition errors, but they both come at a cost: they make dialogues longer either by only letting the user provide small amounts of information at a time (as in strongly system-directed dialogues), 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 dialogue systems did not also have poor turn-taking capabilities. The cost of an additional turn for artificial conversational agents, in time spent and disruption of the conversation flow, is much higher than for human-human conversation. As pointed out in recent publications [6, 7], this weakness comes from the fact that research in spoken dialogue systems, while focusing on high-level concerns such as natural language understanding and dialogue planning, has to a large extent neglected low-level interaction.

Even as more and more complex tasks have been addressed, low-level interaction processes, such as turn-taking, have stayed by and large unchanged. Most systems use a pipeline architecture where the user's speech gets sequentially split into utterances, recognized, parsed, fed to a dialogue manager which produces a response that gets verbalized into natural language and synthesized. In this framework, each component either waits for the previous one to finish before starting its own processing or, as is often the case for the dialogue manager, works asynchronously from the real world, without feedback from it. This approach has some software engineering advantages: it is simple to build and existing components (speech recognizer, parser, etc) can be used as-is and chained together. On the other hand, systems built on a pipeline architecture lend themselves to various interactional problems, such as inappropriate delays [7], spurious interruptions, and turn over-taking (when the user and the system get "out of sync" [6]). A dialogue system architecture that allows real-time processing and reaction is thus essential for better interaction. To this end, multi-layers architectures have been recently proposed for conversational agents [8, 9, 10]. These architectures, inspired by work on

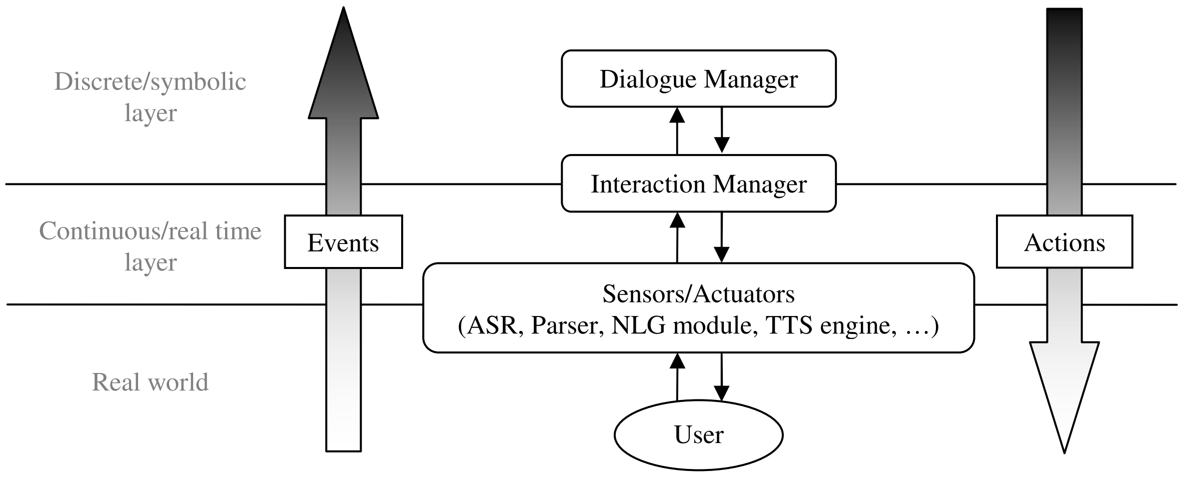

Figure 1. Overview of the proposed architecture

autonomous robots, separate long-term deliberative behavior, including dialogue planning, task modeling and grounding, from immediate reactive behavior such as turn taking. In this paper, we present a new architecture that adopts this layered approach, while making use of the concept of conversational floor to synchronize the layers, thus allowing the representation of the dialogue state maintained by the high-level dialogue manager to constantly match the actual state of the dialogue. The key aspects of the architecture are given in the next section, while Section 3 describes its implementation as Olympus 2, a new version of the Olympus architecture [11] and Section 4 presents the application of Olympus 2 to the Let's Go system, a deployed, telephone-based, bus schedule information system. Section 5 discusses other multi-layer spoken dialogue system architectures, while Section 5 concludes the paper.

2. PROPOSED ARCHITECTURE

2.1. Overview

Conceptually, our architecture distinguishes three layers. 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 lowest layer represents the real world (e.g. the user speaking to interrupt the system). The intermediate layer is a first level of abstraction, which consists of real-time events and actions with continuous properties (e.g. the exact timing and duration of a user utterance, as perceived by the Voice Activity Detector and speech recognizer). Finally, the top layer is the domain of purely symbolic events and actions with typically discrete properties (e.g. a representation of the fact that the user barged in after hearing a specific phrase uttered by the system). The core components of the architecture perform two types of tasks: 1) they accept events and actions at one level and produce events and actions at the next level (event composition/action decomposition), and 2) they produce actions at a certain level in response to events at the same level (control). The interface between the real world and the intermediate layer is achieved by a set of sensors and actuators. No control happens at this level. The interface between the intermediate and top layers is performed by a new 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 Dialogue Manager (DM) plans high-level actions based on high-level events. Being at the top of the architecture, the DM does not perform any composition/decomposition.

2.2. Conversational Floor and Dialogue Management

The role of the dialogue manager (DM) in a dialogue system is two-fold. First it monitors the intentions and beliefs of the participants, as well as the current focus of the conversation. In other words, it keeps track of the (high-level) dialogue state. Second, the DM plans the system's contributions to the conversation[1]. We assume that this latter role is performed by a planning module, which captures the (projected) high-level structure of the dialogue, and an execution module, which sends actions to be executed to lower layers, and monitors events. We do not make any further assumptions on the internals of the planning module. In particular, it could follow any of the common dialogue management formalisms, from finite-state networks, to form filling, to plan-based dialogue management. Many dialogue managers assume that both planning and execution can be performed asynchronously from the real-world. For example, consider the dialogue extract from Figure 3 and the corresponding

[1] Here and thereafter, we use the word "plan" and "planning" in a very broad sense, including for systems that do not use traditional AI planning formalisms (e.g. finite-state or form-based dialogue managers).

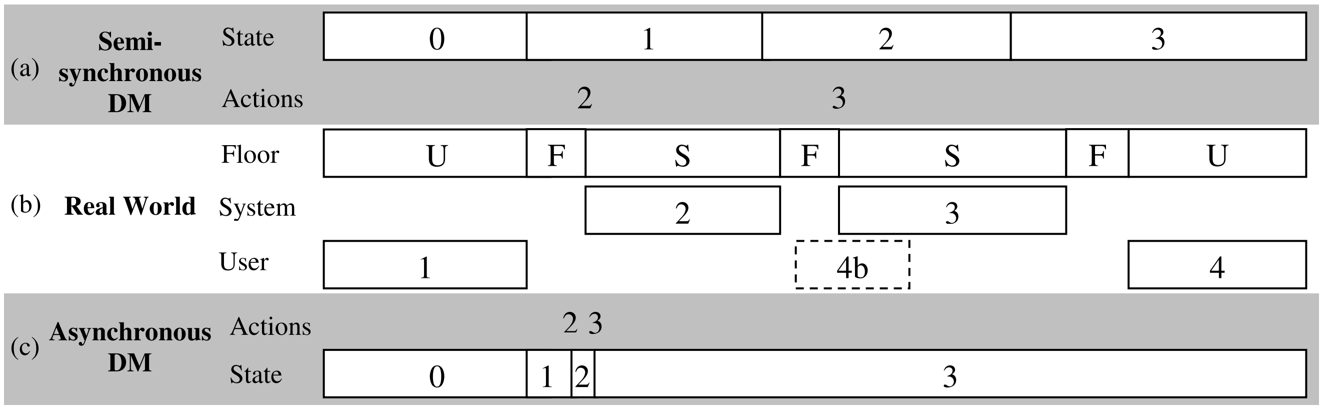

Figure 2. Real-world DM timelines of the dialogue extract from Figure 3. State numbers refer to the dialogue state after the corresponding utterances has been spoken/understood.

…		(0)
User:	I want to go to Boston.	(1)
System:	Going to Boston.	(2)
System:	Do you need a return trip?	(3)
User:	Yes.	(4)

Figure 3. Extract from a dialogue in the flight reservation domain.

timeline from Figure 2. A typical asynchronous DM, as shown in Figure 2 (c), assumes that actions are completed as soon as they have been sent to the lower levels, updates its state accordingly and starts the execution of the next action immediately (e.g. action 3's execution is launched immediately after 2). In such systems, it is up to the actuators to guarantee that utterances are spoken sequentially (i.e. to start utterance 3 after utterance 2 has been spoken). In theory, without execution monitoring, the DM could continue executing future actions until it reaches the (projected) end of the dialogue, without ever waiting for the user to respond. To avoid this, practical DMs resort to synchronization mechanisms extraneous to the planning and execution model (e.g. the Input Pass in RavenClaw [12], or "a separate layer of discourse" in COLLAGEN [13]), which freeze DM execution when user input is expected (e.g. after the DM executes a "question" action).

While asynchronous DMs present the advantage of allowing the system to start synthesizing the next utterance while speaking it, such approaches have two serious limitations. First, the internal state of the DM does not always match that of the actual dialogue (and presumably of the user). This is an issue when conversational (e.g. interruptions and backchannels) and non-conversational (e.g. notifications of change in the environment of a mobile robot) events occur during a system utterance. For instance, in the example of Figure 2, imagine that utterance 4 was spoken by the user not after utterance 3, but after utterance 2 (utterance 4b in Figure 3). An asynchronous DM would still, erroneously, interpret it in state 3, as an answer to the yes/no question. However, given its timing, utterance 4b would better be interpreted as a backchannel response to the implicit confirmation 2. The second issue with asynchronous DMs is that because the DM is on hold while waiting for user responses, no execution can occur until either the user responds or a timeout is triggered. During those waiting phases, the DM cannot handle non-conversational events, which could have conversational consequences (e.g. the system might need to inform the user of a change in the real world).

To address these issues, we introduce the concept of conversational floor into the execution module of the DM. The floor is an additional dialogue state variable that can take three values: `user`, `system`, and `free`. The value of the floor is not decided by the DM but acquired from lower-level modules. Each action that the DM can plan has two markers: one indicates the value(s) in which the floor can be for this action to be executed; the other indicates the value of the floor after the execution of the action is completed. Typically, conversational acts require the floor to be `free`, with the exception of backchannel conversational acts and interruptions. Non-conversational actions (e.g. interacting with a backend database) also do not have floor requirements. In terms of floor transitions, the general behavior is for the floor to become `user` after questions and `free` after statements. The DM only executes actions whose floor requirements are satisfied. When the floor is either `user` or `system`, the DM is still able to accept events, update the dialogue state, perform planning, and execute non-floor requiring actions.

Both floor transitions and dialogue state updates are triggered by events from the Intermediate Layer, i.e. they reflect changes in the real world precisely when they occur. This allows the DM to interpret events, including interruptions and backchannels, in the right context. Through floor and

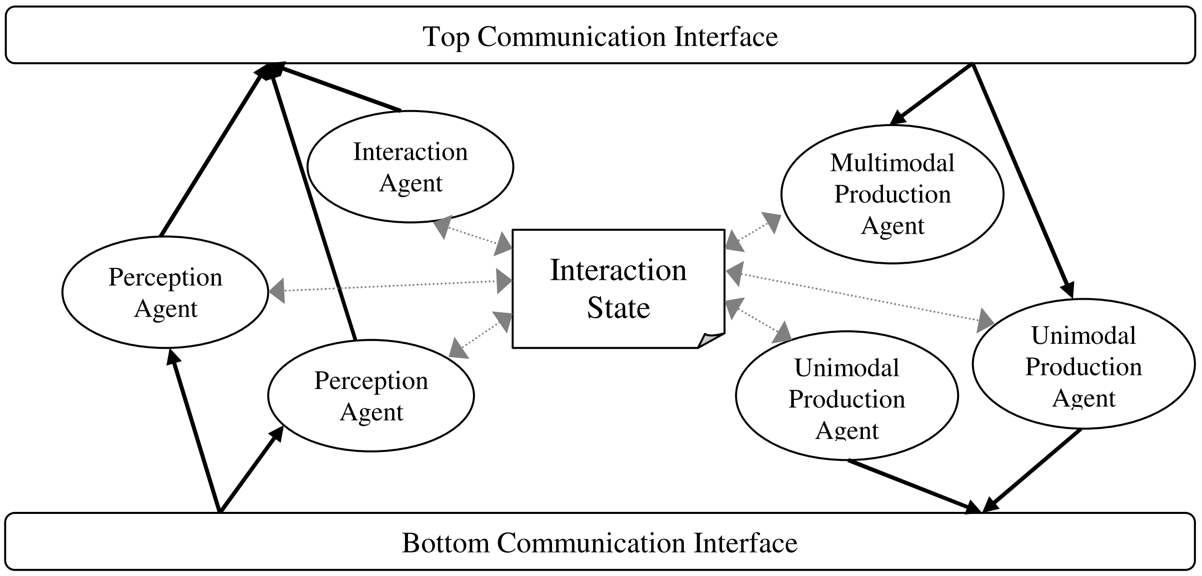
Top Communication Interface
Interaction Agent
Multimodal Production Agent
Perception Agent
Interaction State
Unimodal Production Agent
Perception Agent
Unimodal Production Agent
Bottom Communication Interface

Figure 4. Internal organization of the Apollo Interaction Manager (solid arrows give the flow of events and actions, dotted arrows represent access to the interaction state)

state update events, the execution module of the DM is thus synchronized with the real-world dialogue. The combination of an asynchronous planning module with a synchronous execution module is the essence of a semi-synchronous dialogue manager, whose behavior is illustrated in Figure 2 (a).

2.3. Interaction Management

The Interaction Manager (IM) acts both as the interface between the Intermediate and Top layers, and as the controller of the system's reactive behavior. In particular, it sends appropriate dialogue state and floor update events to the DM. In order to achieve these goals, the IM should be able to:

1. react to Top level actions, Intermediate level events, and timing phenomena
2. integrate a variety of modalities and sensor/actuator types
3. operate in real time

We designed an IM that fulfills these requirements using the architecture illustrated in Figure 4. Two interfaces handle communication of actions and events with the DM (top communication interface), and the sensors and actuators (bottom communication interface). Between these two lies a set of agents, each of which handles unimodal or multimodal perception or production.

All agents can read and modify a common blackboard object that describes the dynamic interaction state. When the IM receives an intermediate-level event from the sensors, the bottom communication interface sends it to the appropriate agent. Based on this event, perception agents update the interaction state. Multimodal integration agents combine information from several unimodal perception agents to update multimodal state variables. Both perception and integration agents can also send high-level events to the DM.

Similarly, when the IM receives an action from the DM, the top communication interface sends it to the appropriate unimodal or multimodal production agent. Production agents update the state and can send intermediate-level actions to the actuators. In addition to the events it receives from the sensors, the bottom 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 can be developed independently and later combined, as long as they share the same definition of the interaction state. While the use of the blackboard guarantees that any agent has access to information from all the other agents, it is easy to allow 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 top and bottom 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.

3. IMPLEMENTATION: OLYMPUS 2

We implemented the proposed architecture as a new version of the Olympus [11] spoken dialogue framework. By doing so we were able to reuse a number of modules and ensure the task-independence of our implementation.

The DM is based on the RavenClaw 2 dialogue management framework, which extends RavenClaw [12] to handle generic actions and events and incorporate the notion of floor as described in section 2.2. For Interaction Management, we created a new module, named Apollo, based on the principles given in section 2.3. Olympus 2 has similar features to its predecessor. It is still highly modular, flexible, transparent, and open-source. These features make the Olympus 2 architecture a suitable platform for research on high-level dialogue management (as was already Olympus), low-level interaction management, as well as on the interaction between these two levels.

4. APPLICATION TO THE LET'S GO SYSTEM

In order to test our approach on an actual system, we ported the Let's Go bus information system to Olympus 2. Let's Go is a publicly available telephone-based system that provides bus schedule information for the Pittsburgh metropolitan area. It was originally built on the Olympus 1 architecture and deployed in 2005 to receive calls from the general public at times when the transit operator's customer service phone line are not manned. In its first two years of operation, Let's Go handled more than 34000 calls and was progressively improved to reach a dialogue completion rate of 76.7%[2].

In April 2007, we ported the system to the new Olympus 2 architecture. 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 and one production agents: the Listener Agent, which handles the ASR/NLU sensor and the Speaker Agent, which handles the NLG/TTS actuator. The turn-taking rules within the agents were hand-written so as to follow a standard behavior, similar to that of Olympus-I. Thus, for example, the system considers that the user yields the floor based on pauses of more than 800ms. 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. As of July 1st, 2007, after three months of operation, the new version of Let's Go has handled 5000 dialogues, 3738 of which have 4 user turns or more. The completion rate among these longer dialogues is 76%, almost identical to the rate in the three months preceding the switch to the new version (the difference is not statistically significant). Similarly, the average duration and number of turns per dialogue have remained stable (resp. from 137.3 s to 138.8 s, and from 15.9 turns to 16.2 turns).

5. DISCUSSION

Other multi-layer approaches to dialogue management have been proposed. An early and important work is that of Thorisson [8, 9]. His model is divided in three layers: the Content Layer, which deals with topics and tasks, the Process Control Layer, which deals with typical dialogue 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). 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 keeping 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 dialogue management. Therefore, it remains unclear how this model would work in the context of complex, task-oriented dialogues. More recently, in [10], Lemon et al propose an architecture for task-oriented dialogue systems that distinguishes a Content Layer and an Interaction Layer. The Content Layer has at its core a Dialogue 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 [10] seems to be exclusively contained in the Interaction Layer and it is not clear how the Dialogue 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 dialogue management framework) work. Focus on a clear distinction between asynchronous planning and synchronous execution is a key component of this work. In particular, our dialogue management model makes explicit how the theoretical concept of conversational floor influences the execution of the dialogue plan. This particularity constitutes a departure from the pure asynchronous models previously proposed. We believe that our model better accounts for the influence of asynchronous events on the plan, as well as for interactions between the lower and higher levels of communication.

Another difference is that we define our layers in terms of level of abstraction, rather than in terms of processing. In this view, software components (except the DM which is at the top), lie at the interface of two layers rather than within one layer. The role of the sensors, actuators and of the IM is

[2] Dialogue completion rate, defined as the proportion of dialogues in which the system provides a result to the user, is an automatic estimate of success rate.

therefore to "translate" events and actions between different levels of abstraction[3].

However, beyond these conceptual differences, we believe our main contribution is to provide the community with an open framework based on the multi-layer approach and to show the applicability of this approach to deployed systems. The fact that Let's Go was ported to Olympus 2 with only minimal modifications to the domain-specific parts of the system confirms that systems can be built on top of reactive architectures without significant overhead in terms of system design. In the near future, we plan to port other existing Olympus-based systems to Olympus 2, as well as to develop new systems. This 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. In the meantime, we will improve the internal model of the IM to better handle turn-taking phenomena such as smooth transitions and interruptions.

6. CONCLUSION

We introduced a new, multi-layer architecture to build task-oriented spoken dialogue systems. Implemented as a new version of the Olympus architecture, it features a new version of the RavenClaw dialogue 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 Dialogue 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.

7. ACKNOWLEDGMENTS

This work is supported by the US National Science Foundation under grant number 0208835, "LET'S GO: improved speech interfaces for the general public". Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would like to thank Dan Bohus, Alex Rudnicky, and many others for their contribution to the original Olympus architecture, as well as Alan W Black for his continuous support and advice. Finally, we would like to acknowledge the Port Authority of Allegheny County for providing access to their database and for their help in making the Let's Go system accessible to Pittsburghers.

[3] This does not mean that there is a one-to-one mapping between events and actions in different layers, just as there is no one-to-one mapping in natural language translation.

8. REFERENCES

[1] O. Lemon, "Context-sensitive speech recognition in isu dialogue systems: results for the grammar switching approach," *CATALOG, 8th Workshop on the Semantics and Pragmatics of Dialogue*, Barcelona, Spain, 2004.

[2] F. Farfán, H. Cuayáhuitl, and A. Portilla, "Evaluating dialogue strategies in a spoken dialogue system for email," *IASTED Artificial Intelligence and Applications*, Manalmádena, Spain, 2003.

[3] J. Edlund, G. Skantze, and R. Carlson, "Higgins - a spoken dialogue system for investigating error handling techniques," *ICSLP 2004*, Jeju, Korea, 2004.

[4] D. Bohus and A. Rudnicky, "Error handling in the RavenClaw dialog management architecture," *HLT/EMNLP 2005*, Vancouver, BC, 2005.

[5] L. Deng and A. Acero, "Challenges in adopting speech recognition", Communications of the ACM, 47:1, pp. 69-75, 2004.

[6] R. Porzel and M. Baudis, "The Tao of CHI: Towards effective human-computer interaction," *HLT/NAACL 2004*, Boston, MA, 2004.

[7] N. Ward, A. Rivera, K. Ward, and D. Novick, "Root causes of lost time and user stress in a simple dialog system," *Interspeech 2005*, Lisbon, Portugal, 2005.

[8] K. R. Thorisson, *Communicative Humanoids: A Computational Model of PsychosocialDialogue Skills*, PhD thesis, Massachusetts Institute of Technology, 1996.

[9] K. R. Thorisson, "Natural Turn-Taking Needs No Manual: Computational Theory and Model, From Perception to Action," *Multimodality in Language and Speech Systems*, pp 173–207. Kluwer Academic Publishers, 2002.

[10] O. Lemon, L. Cavedon, and B. Kelly, "Managing dialogue interaction: A multi-layered approach," *SIGdial Workshop 2003*, Sapporo, Japan, 2003.

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

[12] D. Bohus, and A. Rudnicky, "RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda," *Eurospeech 2003*, Geneva, Switzerland, 2003.

[13] C. Rich, N.B. Lesh, J. Rickel, and A. Garland, "A Plug-in Architecture for Generating Collaborative Agent Responses", International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 782-789, 2002