An Architecture for Dialogue Management, Context Tracking, and Pragmatic Adaptation in Spoken Dialogue Systems

Susann LuperFoy, Dan Loehr, David Duff, Keith Miller, Florence Reeder, Lisa Harper
The MITRE Corporation
1820 Dolley Madison Boulevard, McLean VA 22102 USA
{luperfoy, loehr, duff, keith, freeder, lisah}@mitre.org

Abstract

This paper details a software architecture for discourse processing in spoken dialogue systems, where the three component tasks of discourse processing are (1) Dialogue Management, (2) Context Tracking, and (3) Pragmatic Adaptation. We define these three component tasks and describe their roles in a complex, near-future scenario in which multiple humans interact with each other and with computers in multiple, simultaneous dialogue exchanges. This paper reports on the software modules that accomplish the three component tasks of discourse processing, and an architecture for the interaction among these modules and with other modules of the spoken dialogue system. A motivation of this work is reusable discourse processing software for integration with non-discourse modules in spoken dialogue systems. We document the use of this architecture and its components in several prototypes, and also discuss its potential application to spoken dialogue systems defined in the near-future scenario.

Introduction

We present an architecture for spoken dialogue systems for both human-computer interaction and computer mediation or analysis of human dialogue. The architecture shares many components with those of existing spoken dialogue systems, such as CommandTalk (Moore et al. 1997), Galaxy (Goddeau et al. 1994), TRAINS (Allen et al. 1995), Verbmobil (Wahlster 1993), Waxholm (Carlson 1996), and others. Our architecture is distinguished from these in its treatment of discourse-level processing.

Most architectures, including ours, contain modules for speech recognition and natural language interpretation (such as morphology, syntax, and sentential semantics). Many include a module for interfacing with the back-end application. If the dialogue is two-way, the architectures also include modules for natural language generation and speech synthesis.

Architectures differ in how they handle discourse. Some have a single, separate module labeled “discourse processor”, “dialogue component”, or perhaps “contextual interpretation”. Others, including earlier versions of our system, bury discourse functions inside other modules, such as natural language interpretation or the back-end interface.

An innovation of this work is the compartmentalization of discourse processing into three generically definable components—Dialogue Management, Context Tracking, and Pragmatic Adaptation (described in Section 1 below)—and the software control structure for interaction between these and other components of a spoken dialogue system (Section 2).

In Section 3, we examine the dialogue processing requirement in a complex scenario involving multiple users and multiple simultaneous dialogues of diverse types. We describe how our architecture supports implementations of such a scenario. Finally, we describe two implemented spoken dialogue systems that embody this architecture (Section 4).

1 Component Tasks of Discourse Processing

We divide discourse-level processing into three component tasks: Dialogue Management, Context Tracking, and Pragmatic Adaptation.

1.1 Dialogue Management

The Dialogue Manager is an oversight module whose purpose is to facilitate the interaction between dialogue participants. In a user-initiated system, the dialogue manager directs the processing of an input utterance from one component to another through interpretation and back-end system response. In the process, it detects and handles dialogue trouble, invokes the context tracker when updates are necessary, generates system output, and so on.

Our conception of Dialogue Manager as controller becomes increasingly relevant as the software system moves away from the standard "NL pipeline" in order to deal with dialogue disfluencies. Its oversight perspective affords it (and the architecture) certain capabilities, which are listed in Table 1.

#	*Capability*
1	Supports mixed-initiative system by fielding spontaneous input from either participant and routing it to the appropriate components.
2	Supports non-linguistic dialogue "events" by accepting them and routing them to the Context Tracker (below).
3	Increases overall system performance. For example, awareness of system output allows the Dialogue Manager to predict user input, boosting speech recognition accuracy. Similarly, if the back-end introduces a new word into the discourse, the Dialogue Manager can request the speech recognizer to add it to its vocabulary for later recognition.
4	Supports meta-dialogues between the dialogue system itself and either participant. An example might be a participant's questions about the status of the dialogue system.
5	Acts as a central point for dialogue troubleshooting, after (Duff et al. 1996). If any component has insufficient input to perform its task, it can alert the Dialogue Manager, which can then reconsult a previously invoked component for different output.

Table 1. Dialogue Manager Capabilities

The Dialogue Manager is the primary locus of the dialogue agent's outward personality as a function of interaction style; its simple protocol specifies conditions for interrupting user speech for permitting interruption by the user, when to initiate repair dialogues, and how often to back-channel.

1.2 Context Tracking

The Context Tracker maintains a record of the discourse context which it and other components can consult in order to (a) resolve dependent forms that occur in input utterances and (b) generate appropriate context-dependent forms for achieving natural output. Interpretation of definite pronouns, demonstratives (*this*, *those*), indexicals (*you*, *now*, *here*, *tomorrow*), definite NPs (*a car*...*the car*), *one*-anaphora (*the earlier one*) and ellipsis (*how about Seattle*) all rely on stored context.

The Context Tracker strives to record only those entities and events that could become eligible for reference. Context thus includes linguistic communicative acts (verbalizations), non-linguistic communicative acts (gesture), and non-communicative events that are deemed salient. Since determining salience requires a judgement, our implementations rely on heuristic rules to decide which events and objects get entered into the context representation. For example, the disappearance of a simulated vehicle off the edge of a map display might be deemed salient relative to a particular user model, the discourse history, or the task structure.

1.3 Pragmatic Adaptation

The Pragmatic Adaptation module serves as the boundary between language and action by determining what action to take given an interpreted input utterance or a back-end response. This module's role is to "make sense" of a communicative act in the current linguistic and non-linguistic context.

The Pragmatic Adapter receives an interpretation of an input utterance with context-dependent forms resolved. It then proceeds to translate that utterance into a valid back-end command. It checks for violations of the Domain Model,

which contains information about the back-end system such as allowable parameter values for command arguments. It also checks for commands that are infelicitous given the current Back-end State (e.g., the referenced vehicle does not exist at the moment). The Pragmatic Adapter combines the result of these simple tests and a set of if-then heuristics to determine whether to send through the command or to intercept the utterance and notify the Dialogue Manager to initiate a repair dialogue with the user.

The Pragmatic Adapter receives output responses from the back-end and adapts or "translates" them into natural language communications which get incorporated by the Context Tracker into the dialogue history.

2 An Architecture for Spoken Dialogue Systems

Having introduced our three discourse components, we now present our overall architecture. It is laid out in Figure 1, and its components are described in Table 2, starting from the user and going clockwise. The discourse components are left in white, while non-discourse components have been shaded gray.

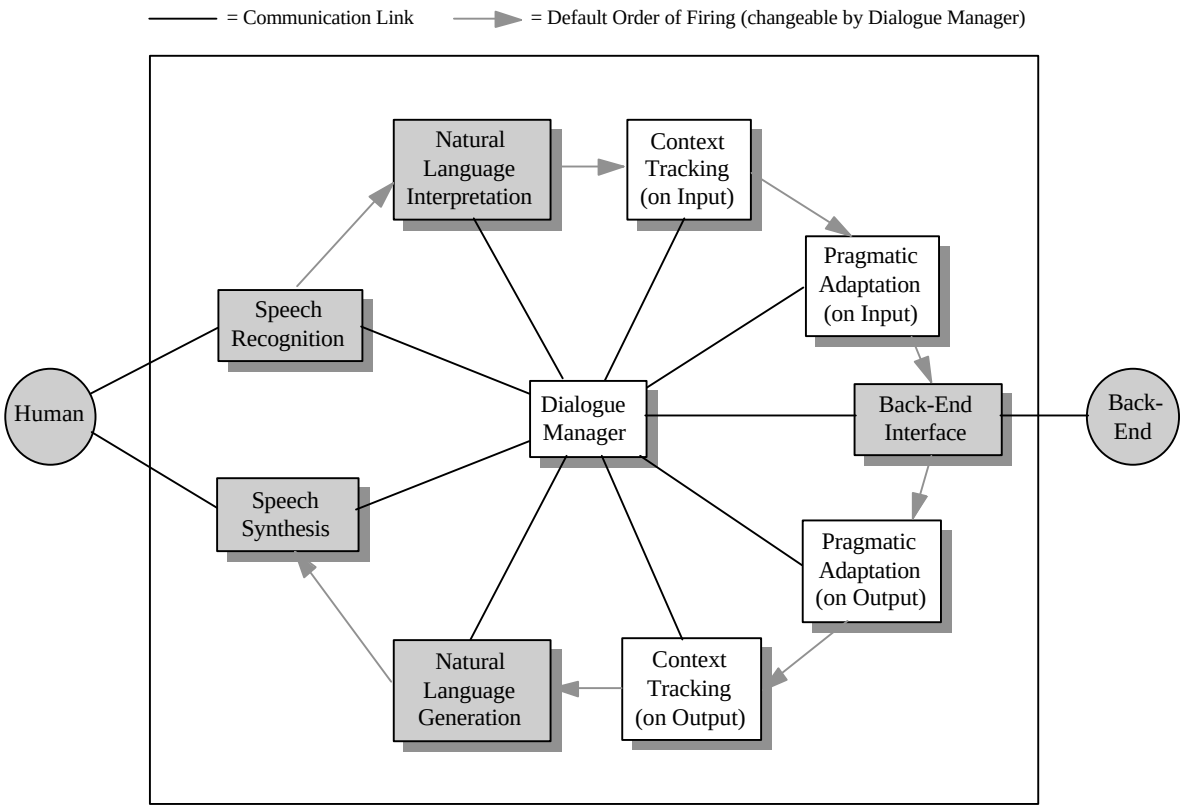

Figure 1. An Architecture for Spoken Dialogue Systems, with Discourse Components in White

Component (Agent)	*Brief Description*	*Possible Input*	*Possible Output*
Speech Recognition	Convert waveform to string of words	Waveform	Text string
NL Interpretation	Convert words to meaning representation	Text string	Logical form
Context Tracking	Track discourse entities of input utterance,	Logical form (with	Logical form (with de-

on Input	resolve dependent references	dependent references)	pendent references replaced by their referents)
Pragmatic Adaptation on Input	Convert logical form to back-end command	Logical form	Back-end command
Back-End Interface	Layer on back-end application to allow communication with Dialogue Manager	Back-end command	Back-end response
Pragmatic Adaptation on Output	Convert back-end response to logical form representation of communicative act	Back-end response	Logical form
Context Tracking on Output	Track discourse entities of output utterance, insert dependent references (if desired)	Logical form (w/out dependent references)	Logical form (conditioned by discourse context)
NL Generation	Convert meaning representation to words	Logical form	Text string
Speech Synthesis	Convert words to waveform	Text string	Waveform
Dialogue Manager	High-level control, intelligently route information between all agents and participants (see section 1.1) based on its own protocol for interaction.	Various	Various

Table 2. Description of the Architecture Components, with Discourse Components in White

Several items are of note in Figure 1 and Table 2. First, although a default firing order is shown, this order is perturbed any time dialogue trouble arises. For example, a Speech Recognition (SR) error, may be detected after Natural Language Interpretation fails to parse the output of SR. Rather than continuing the flow on towards the back-end, the Dialogue Manager can re-consult SR for other hypotheses. Alternatively, the Dialogue Manager can fire Natural Language Generation with an output request for clarification. That request gets incorporated into the context representation by Context Tracking, the dialogue state is "pushed" in a repair dialogue, and a string is ultimately sent to Speech Synthesis for delivery to the user's ear. The next utterance is then interpreted in the context of the repair dialogue.

Note also that Context Tracking and Pragmatics Adaptation are called twice each: on "input" (from the user), and on "output" (from the back-end). The logical Context Tracker may be implemented as one or as two related modules, together tracking both sides of that dialogue so that either user or system can make anaphoric mention of entities introduced earlier.

3 A Near-Future Scenario of Spoken Dialogue Systems

3.1 The Scenario

We build on images from the popular science fiction series Star Trek as a rich source of dialogue types in complex interrelations. These example dialogues have more primitive cousins under development today.

Briefly, our example dialogue types are listed in Table 3.

Type	*Metaphor*	*Example*	*Participant A*	*Participant B*	*Participant C*
Dialogue with an *Appliance*	Food Replicator	The "Food Replicator" on Star Trek accepts structured English command language such as "Tea. Earl Grey. Hot" and produces results in the physical world.	Human	Food Replicator	
Dialogue with an *Application*	Ship's Computer	The ship's computer on Star Trek is an advanced application which can understand natural language queries, and replies either via actions or via a multimodal interface.	Human	Ship's Computer	

Dialogue with an Intelligent *Robot*	Android "Data"	"Data" on Star Trek converses as a human while providing information processing of a computer and is capable of action in the physical world.	Human	Android "Data"	
Computer *Mediation* of Human Dialogue	Universal Translator	Star Trek's "Universal Translator" is capable of automatically interpreting between any two humans	Human	Human	Universal Translator
Computer *Analysis* of Human Dialogue	Conversation Playback	The ship's computer has the ability to retrieve, play back, and analyze previously-recorded conversations. In this sense, the dialogue becomes empirical data to be analyzed.	Human	Human	Ship's Computer
Dialogue between *2 characters*	Holodeck	Star Trek's "Holodeck" creates simulated humans (or *characters*) as actors, for the entertainment or training of human viewers.	Character	Character	

Table 3. A Scenario of Dialogue Types

3.2 Application of the Architecture to the Scenario

We now describe the role our architecture, and specifically our discourse components, play in these near-future examples.

3.2.1 Dialogue with a Back-End Computer

The first three examples illustrate dialogues in which a human is talking to a computer. One dimension distinguishing the three examples is the agent's intelligent use of context. In a dialogue with an "appliance", simple, structured, unambiguous command language utterances are interpreted one at a time in isolation from prior dialogue history. The Pragmatic Adaptation facility can follow a simple scheme for mapping each utterance to one of a very few back-end commands. The Context Tracker has no cross-sentence dependent references to contend with, and finally, since the appliance provides no linguistic feedback, the Dialogue Manager fires none of the "output" components (from back-end to human). In a dialogue with more sophisticated application or with a robot, the Dialogue Manager, Context Tracker, and Pragmatic Adapter need greater functionality, to handle both linguistic and non-linguistic events in both directions.

3.2.2 Computer-Mediated Dialogue

The fourth example, that of the Universal Translator, is representative of a general dialogue type we label *Mediator*, in which an agent plays a mediation role between humans. In addition to interpretation, other roles of the mediator might be (Table 4):

#	*Mediator Role*
1	A *Genie*, which is available for meta-dialogues with the system itself, instead of with the dialogue partner (much as a human might ask an interpreter to repeat the partner's last utterance).
2	*A Moderator*, which, in multi-party dialogues, enforces an agreed-upon interaction protocol, such as Robert's Rules of Order or a talk-show format (under control of the host).
3	*A Bouncer*, which decides who may join the dialogue based on current enrollment (first-come-first-served), clearance level, invitation list, etc., as well as permitting different types of participation, so that some may only listen while others may fully participate.
4	*A Stenographer*, which records the dialogue, and prepares a "visualization" of the dialogue structure.

Table 4. Roles of a Mediator Agent

Our architecture is applicable to mediated dialogues as well. In fact, it was first developed for bilingual dialogue in a voice-to-voice machine translation application. In this application, the Dialogue Manager is available for meta-dialogues with either user (as in *Could you repeat her last utterance?*), and the Context Tracker can use a single discourse representation structure to track the unfolding context in both languages.

3.2.3 Computer-Analyzed Dialogue

Our fifth example, a post-hoc analysis of a dialogue, does not require real-time processing. It is,

nonetheless, a dialogue which can be analyzed using the components of our architecture, exactly as if it were real-time. The only difference is that no generation will be required, only analysis; thus, the Dialogue Manager need only fire the "input" components on each utterance.

3.2.4 Character-Character Dialogue

Our last example concerns a simulated human dialogue between two computer characters, for the benefit of human viewers. Such character-character dialogues have been produced by several researchers, including (Kalra et al. 1998). Here, the architecture applies at two levels. First, the architecture can be internal to each agent, to implement that agent's conversational ability. Second, the architecture can be used externally to analyze the agents' dialogue, as discussed in the previous section.

4 Implementations of the Architecture

We have implemented two spoken dialogue systems using the architecture presented. The first is a telephone-based interface to a simulated employee Time Reporting System (TRS), as might be used at a large corporation. We then ported the system to a spoken interface to a battlefield simulation (Modular Semi-Automated Forces, or ModSAF).

In our implementation of this architecture, each component is a unique agent which may reside on its own platform and communicate over a network. The middleware our agents use to communicate is the Open Agent Architecture (OAA) (Moran et al. 1997) from SRI. The OAA's flexibility allowed us to easily hook up modules and experiment with the division of labor between the three discourse components we are studying. We treat the Dialogue Manager as a special OAA agent that insists on being called frequently so that it can monitor the progress of communicative events through the system.

4.1 The Time Reporting System (TRS)

The architecture components in our TRS system are listed in Table 5, along with their specific implementations used. Each implemented module included a thin OAA agent layer, allowing it to communicate via the OAA.

Component	*Implementation of Component*
Speech Recognition	Hark™, (BBN)
Speech Synthesis	TrueTalk (Entropic)
NL Interpretation/Generation	Simulated
Back-End Interface	Simulated
Context Tracking	(LuperFoy 1992)
Pragmatic Adaptation	Currently Simulated
Dialogue Manager	Current Development

Table 5. Components of TRS System, with Discourse Components in White

Components not in our focus (shaded in gray) are either commercial or simulated software. For Context Tracking, we use an algorithm based on (LuperFoy 1992). For Dialogue Management, we developed a simple agent able to control a system-initiated dialogue, as well as handle non-linguistic events from the back-end. The third discourse component, Pragmatic Adaptation, awaits future research, and was simulated for this system.

Figure 2 presents a sample TRS dialogue.

System: Welcome. What is your employee number?
User: 12345
System: What is your password?
User: 54321
System: How can I help you?
User: What's the first charge number?
System: 123GK498J
User: What's the name of that task?
System: Project X
User: Charge 6 hours to it today for me.
System: 6 hours has been charged to Project X.

Figure 2. Sample TRS Dialogue

When the user logs in, the back-end system brings up a non-linguistic event—the list of tasks, with associated charge numbers, which belong to the user. The Dialogue Manager receives this and passes it to the Context Tracker. The Context Tracker is then able to resolve *the first charge number*, as well as subsequent dependent references such as *that task*, *it*, and *today*.

4.2 The ModSAF Interface

We ported the TRS demo to a simulated battlefield back-end called ModSAF. We used the same components with the exception of the speech recognizer and the back-end interface. The Dialogue Manager was improved over the TRS demo in several ways. First, we added the capability of the Dialogue Manager to dynamically inform the speech recognizer of what input to expect, i.e., which language model to use. The Dialogue Manager could also add words to the speech recognizer's vocabulary on the fly. We chose Nuance (from Nuance Communications) as our speech recognition component specifically because it supports such run-time updates.

Figure 3 presents a sample ModSAF dialogue. Note that only the user speaks.

- Create an M1A2 platoon.
- Name it Bravo.
- Give it location 4 9 degrees 3 0 minutes north, 1 1 degrees 4 5 minutes east.
- Bravo, advance to checkpoint Charlie.

(At this point, a new platoon appears on the screen, created by another player in the simulation)

- Zoom in on that new platoon.
- Bravo, change location and approach X.

(Where X is the name of the new platoon.)

Figure 3. Sample ModSAF Dialogue

When the user asks to create an entity, the Dialogue Manager detects the beginning of a subdialogue, and informs the speech recognizer to restrict its expected grammar to that of entity creation (name and location). Later, the back-end (ModSAF) sends the Dialogue Manager a non-linguistic event, in which a different platoon (created by another player in the simulation) appears. This event includes a name for the new platoon; the Dialogue Manager passes this to the speech recognizer, so that it may later recognize it. In addition, the event is passed to the Context Tracker, so that it may later resolve the reference *that new platoon*.

To illustrate some advantages of our architecture, we briefly mention what we needed to change to port from TRS to ModSAF. First, the Context Tracker needed no change at all—operating on linguistic principles, it is domain-independent. LuperFoy's framework does provide for a layer connected to a knowledge source, for external context—this would need to be changed when changing domains. The Dialogue Manager also required little change to its core code, adding only the ability to influence the speech recognizer. The Pragmatic Adaptation Module, being dependent on the domain of the back-end, is where most changes are needed when switching domains.

Conclusion

We have presented a modular, flexible architecture for spoken dialogue systems which separates discourse processing into three component tasks with three corresponding software modules: Dialogue Management, Context Tracking, and Pragmatic Adaptation. We discussed the roles of these components in a complex, near-future scenario portraying a variety of dialogue types. We closed by describing implementations of these dialogues using the architecture presented, including development and porting of the first two discourse components.

The architecture itself is derived from a standard blackboard control structure. This is appropriate for our current dialogue processing research in two ways. First, it does not require a prior full enumeration of all possible subroutine firing sequences. Rather, the possibilities emerge from local decisions made by modules that communicate with the blackboard, depositing data and consuming data from the blackboard. Second, as we learn categories of dialogue segment types, we can move away from the fully decentralized control structure, to one in which the central Dialogue Manager, as a blackboard module with special status, assumes increasing decision power for processing flow, in cases of dialogue segment type with which it is familiar. The intended contribution of this work is thus in the generic definition of standard dialogue functions such as dynamic troubleshooting (repair), context updating, anaphora resolution, and translation of natural language interpretations into functional inter-

face languages of back-end systems.

Future work includes investigation of issues raised when a human is engaged in more than one of our scenario dialogues concurrently. For example, how does one speech enabled dialogue system among many determine when it is being addressed by the user, and how can the system judge whether the current utterance is human-computer, i.e., to be fully interpreted and acted upon by the system as opposed to a human-human utterance that is to be simply recorded, transcribed, or translated without interpretation.

References

Allen J., Schubert L., Ferguson G., Heeman P., Hwang C., Kato T., Light M., Martin N., Miller B., Poesio M., Traum D. (1995) *The TRAINS Project: A case study in building a conversational planning agent.* Journal of Experimental and Theoretical AI, 7, pp. 7—48.

Carlson R. (1996) *The Dialogue Component in the Waxholm System.* Proc. Twente Workshop on Language Technology: Dialogue Management in Natural Language Systems, University of Twente, the Netherlands.

Duff D., Gates B., LuperFoy S. (1996) *A Centralized Troubleshooting Mechanism for a Spoken Dialogue Interface to a Simulation Application.* Proc. International Conference on Spoken Language Processing.

Goddeau D., Brill E., Glass J., Pao C., Phillips M., Polifroni J., Seneff S., Zue V. (1994) *GALAXY: A Human-Language Interface to On-line Travel Information.* Proc. International Conference on Spoken Language Processing.

Kalra P., Thalmann N., Becheiraz, P., Thalmann D. (1998) *Communication Between Synthetic Actors.* In "Automated Spoken Dialogue Systems", S. LuperFoy, ed. MIT Press (forthcoming).

LuperFoy. S (1992) *The Representation of Multimodal User-Interface Dialogues Using Discourse Pegs.* Proc. Annual Meeting of the Association for Computational Linguistics.

Moore R., Dowding J., Bratt H., Gawron J., Gorfu Y., Cheyer, A. (1997) *CommandTalk: A Spoken-Language Interface for Battlefield Simulations.* Proc. Fifth Conference on Applied Natural Language Processing.

Moran D., Cheyer A., Julia L., Martin D. Park S. (1997) *Multimodal User Interfaces in the Open Agent Architecture.* Proc. International Conference on Intelligent User Interfaces.

Wahlster W. (1993) *Verbmobil: Translation of Face-To-Face Dialogues.* In "Grundlagen und Anwendungen der Künstlichen Intelligenz", O. Herzog, T. Christaller, D. Schütt, eds., Springer.

Page: 3

[DAD1] I'm not completely sure I understand this, but I tried to make it make sense.

Page: 5

[DAD2] Every row in table 3 corresponds directly to a column in table 4. It seems that these two tables should be combined, or at least, one of them should be transposed (probably table 4) so as not to obscure this connection.

Page: 6

[DAD3] I found this part confusing. A reader might thhat you are talking about **implementing** dialogue between two compujter characters, but you are only talking about analyzing it.