An Architecture For More Realistic Conversational Systems

James Allen, George Ferguson, Amanda Stent
Department of Computer Science
University of Rochester
Rochester, NY 14627-0226
{james,ferguson,stent}@cs.rochester.edu
http://www.cs.rochester.edu/research/trips/

ABSTRACT

In this paper, we describe an architecture for conversational systems that enables human-like performance along several important dimensions. First, interpretation is incremental, multi-level, and involves both general and task- and domain-specific knowledge. Second, generation is also incremental, proceeds in parallel with interpretation, and accounts for phenomena such as turn-taking, grounding and interruptions. Finally, the overall behavior of the system in the task at hand is determined by the (incremental) results of interpretation, the persistent goals and obligations of the system, and exogenous events of which it becomes aware. As a practical matter, the architecture supports a separation of responsibilities that enhances portability to new tasks and domains.

Keywords

Architectures for intelligent, cooperative, distributed, and multimodal interfaces; conversational systems

1. INTRODUCTION

Our goal is to design and build systems that approach human performance in conversational interaction. We limit our study to *practical dialogues*: dialogues in which the conversants are cooperatively pursuing specific goals or tasks. Applications involving practical dialogues include planning (e.g. designing a kitchen), information retrieval (e.g. finding out the weather), customer service (e.g. booking an airline flight), advice-giving (e.g. helping assemble modular furniture) or crisis management (e.g. a 911 center). In fact, the class of practical dialogues includes almost anything about which people might want to interact with a computer.

TRIPS, The Rochester Interactive Planning System [6], is an end-to-end system that can interact robustly and in near real-time using spoken language and other modalities. It has participated successfully in dialogues with untrained users in several different simple problem solving domains. Our experience building this system, however, revealed several problems that motivated the current work.

1.1 Incrementality

Like most other dialogue systems that have been built, TRIPS enforces strict turn taking between user and system, and processes each utterance sequentially through three stages: interpretation–dialogue management–generation. Unfortunately, these restrictions make the interaction unnatural and stilted, and will ultimately interfere with the user's ability to focus on the problem itself rather than on making the interaction work. We want an architecture that allows a more natural form of interaction; this requires incremental understanding and generation with flexible turn-taking.

Here are several examples of human conversation that illustrate some of the problems with processing in stages. All examples are taken from a corpus collected in an emergency management task set in Monroe County, NY [17]. Plus signs (+) denote simultaneous speech, and "␣" denotes silence.

First, in human-human conversation participants frequently ground (confirm their understanding of) each other's contributions using utterances such as "okay" and "mm-hm". Clearly, incremental understanding and generation are required if we are to capture this behavior. In the following example, A acknowledges each item in B's answers about locations where there are road outages.

Excerpt from Dialogue s16

A: can you give me the first uh ␣ outage
B: okay
B: so Elmwood bridge
A: okay
B: um ␣ Thurston road
A: mm-hm
B: + Three Eighty + Three at Brooks
A: + okay +
A: mm-hm
B: and Four Ninety at the inner ␣ loop
A: okay

Second, in human-human dialogues the responder frequently acknowledges the initiator's utterance immediately after it is completed and before they have performed the tasks they need to do to fully respond. In the next excerpt, A asks for problems other than road outages. B responds with an immediate acknowledgment. Evidence of problem solving activity is revealed by B smacking their lips ("lipsmack") and silence, and then B starts to respond to the request.

Excerpt from Dialogue s16

A: and what are the ␣ other um ␣ did you have just beside road $+_1$ outages $+_1$

B: $+_1$ okay $+_1$ <lipsmack> ␣ um Three Eighty Three and Brooks $+_2$ ␣ is $+_2$ a ␣ road out ␣ and an electric line down

A: $+_2$ Brooks mm hm $+_2$

A: okay

A sequential architecture, requiring interpretation and problem solving to be complete before generation begins, cannot produce this behavior in any principled way.

A third example involves interruptions, where the initiator starts to speak again after the responder has formulated a response and possibly started to produce it. In the example below, B starts to respond to A's initial statement but then A continues speaking.

Excerpt from Dialogue s6

A: and he's going to pull the tree

B: mm hm

B: and + there's mm +

A: + so ␣ he'll be + done at ␣ um ␣ he's going to be done at ␣ in forty minutes

We believe effective conversational systems are going to have to be able to interact in these ways, which are perfectly natural (in fact are the usual mode of operation) for humans. It may be that machines will not duplicate human behavior exactly, but they will realize the same conversational goals using the communication modalities they have. Rather than saying "uh-huh," for instance, the system might ground a referring expression by highlighting it on a display. Note also that the interruption example requires much more than a "barge-in" capability. B needs to interpret A's second utterance as a continuation of the first, and does not simply abandon its goal of responding to the first. When B gets the turn, B may decide to still respond in the same way, or to modify its response to account for new information.

1.2 Initiative

Another reason TRIPS does not currently support completely natural dialogue is that, like most other dialogue systems, it is quite limited in the form of mixed-initiative interaction it supports. It supports taking discourse-level initiative (cf. [3]) for clarifications and corrections, but does not allow shifting of task-level initiative during the interaction. The reason is that system behavior is driven by the dialogue manager, which focuses on interpreting user input. This means that the system's own independent goals are deemphasized. The behavior of a conversational agent should ideally be determined by three factors, not just one: the interpretation of the last user utterance (if any), the agent's own persistent goals and obligations, and exogenous events of which it becomes aware.

For instance, in the Monroe domain, one person often chooses to ignore the other person's last utterance and leads the conversation to discuss some other issue. Sometimes they explicitly acknowledge the other's contribution and promise to address it later, as in the following:

Excerpt from Dialogue s16

A: can you ␣ $+_1$ can $+_1$ you go over the ␣ the thing $+_2$ for me again $+_2$

B: $+_1$ i $+_1$

B: $+_2$ yeah in one $+_2$ minute

B: i have to ␣ clarify at the end here ....

In other cases, they simply address some issue they apparently think is more important than following the other's lead, as in the following example where B does not address A's suggestion about using helicopters in any explicit way.

Excerpt from Dialogue s12

A: we can we ␣ we can either uh

A : i guess we have to decide how to break up ␣ this

A: we can ␣ make three trips with a helicopter

B: so i guess we should send one ambulance straight off ␣ to ␣ marketplace right ␣ now ␣ right

1.3 Portability

Finally, on a practical note, while TRIPS was designed to separate discourse interpretation from task and domain reasoning, in practice domain- and task-specific knowledge ended up being used directly in the dialogue manager. This made it more difficult to port the system to different domains and also hid the difference between general domain-independent discourse behavior and task-specific behavior in a particular domain.

To address these problems, we have developed a new architecture for the "core" of our conversational system that involves asynchronous interpretation, generation, and system planning/acting processes. This design simplifies the incremental development of new conversational behaviors. In addition, our architecture has a clean separation between discourse modeling and task/domain levels of reasoning, which (a) enhances our ability to handle more complex domains, (b) improves portability between domains; and (c) allows for richer forms of task-level initiative.

The remainder of this paper describes our new architecture in detail. The next section presents an overview of the design and detailed descriptions of the major components. A brief but detailed example illustrates the architecture in action. We conclude with a discussion of related work on conversational systems and the current status of our implementation.

2. ARCHITECTURE DESCRIPTION

As mentioned previously, we have been developing conversational agents for some years as part of the TRAINS [7] and TRIPS [6] projects. TRIPS is designed as a loosely-coupled collection of components that exchange information by passing messages. There are components for speech processing (both recognition and synthesis), language understanding, dialogue management, problem solving, and so on.

In previous versions of the TRIPS system, the Dialogue Manager component (DM) performed several functions:

- Interpretation of user input in context

- Maintenance of discourse context
- Planning the content (but not the form) of system responses
- Managing problem solving and planning

Having all these functions performed by one component led to several disadvantages. The distinction between domain planning and discourse planning was obscured. It became difficult to improve interpretation and response planning, because the two were so closely knit. Incremental processing was difficult to achieve, because all input had to pass through the DM (even if no domain reasoning was going to occur, but only discourse planning). Finally, porting the system to new tasks and domains was hampered by the interconnections between the various types of knowledge within the DM.

The new core architecture of TRIPS is shown in Figure 1. There are three main processing components. The Interpretation Manager (IM) interprets user input as it arises. It broadcasts the recognized speech acts and their interpretation as problem solving actions, and incrementally updates the Discourse Context. The Behavioral Agent (BA) is most closely related to the autonomous "heart" of the agent. It plans system behavior based on its goals and obligations, the user's utterances and actions, and changes in the world state. Actions that involve communication and collaboration with the user are sent to the Generation Manager (GM). The GM plans the specific content of utterances and display updates. Its behavior is driven by discourse obligations (from the Discourse Context), and directives it receives from the BA. The glue between the layers is an abstract model of problem solving in which both user and system contributions to the collaborative task can be expressed.

All three components operate asynchronously. For instance, the GM might be generating an acknowledgment while the BA is still deciding what to do. And if the user starts speaking again, the IM will start interpreting these new actions. The Discourse Context maintains the shared state needed to coordinate interpretation and generation.

In the remainder of this section, we describe the major components in more detail, including descriptions of the Discourse Context, Problem Solving Model, and Task Manager.

2.1 Discourse Context

The TRIPS Discourse Context provides information to coordinate the system's conversational behavior. First, it supplies sufficient information to generate and interpret anaphoric expressions and to interpret forms of ellipsis. Given the real-time nature of the interactions, and the fact that the system may have its own goals and receive reports about external events, the discourse context must also provide information about the status of the turn (i.e. can I speak now or should I wait?), and what discourse obligations are currently outstanding (cf. [19]). The latter is especially important when the system chooses to pursue some other goal (e.g. notifying the user of an accident) rather than perform the expected dialogue act (e.g. answering a question); to be coherent and cooperative, the system should usually still satisfy outstanding discourse obligations, even if this is done simply by means of an apology. Finally, as we move towards open-mike interactive systems, we must also identify and generate appropriate

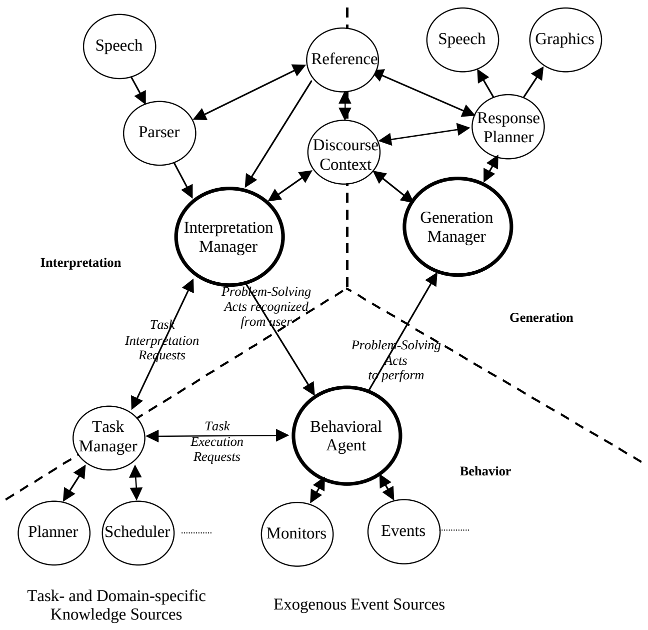

Figure 1: New Core Architecture

grounding behaviors. To support these needs, the TRIPS discourse context contains the following information:

1. A model of the current salient entities in the discourse, to support interpretation and generation of anaphoric expressions;
2. The structure and interpretation of the immediately preceding utterance, to support ellipsis resolution and clarification questions;
3. The current status of the turn—whether it is assigned to one conversant or currently open.
4. A discourse history consisting of the speech-act interpretations of the utterances in the conversation so far, together with an indication of which utterances have been grounded;
5. The current discourse obligations, typically to respond to the other conversant's last utterance. Obligations may act as a stack during clarification subdialogues, or short-term interruptions, but this stack never becomes very large.

This is a richer discourse model than found in most systems (although see [12] for a model of similar richness).

2.2 Abstract Problem Solving Model

The core modules of the conversational agent, the IM, BA and GM, use general models of collaborative problem solving, but these models remain at an abstract level, common to all practical dialogues. This model is formalized as a set of actions that can be performed on problem solving objects. The problem solving objects include *objectives* (goals being pursued), *solutions* (proposed courses of action or structures that may achieve an objective), *resources* (objects used in solutions, such as trucks for transportation, space in kitchen design), and *situations* (settings in which solutions are used to attain objectives).

In general, there are a number of different actions agents can perform as they collaboratively solve problems. Many of these can apply to any problem solving object. For example, agents may *create* new objectives, new solutions, new situations (for hypothetical reasoning) and new resources (for resource planning). Other actions in our abstract problem solving model include *select* (e.g.focus on a particular objective), *evaluate* (e.g. determine how long a solution might take), *compare* (e.g. compare two solutions to the same objective), *modify* (e.g. change some aspect of a solution, change what resources are available), *repair* (e.g. fix an old solution so that it works) and *abandon* (e.g. give up on an objective, throw out a possible solution)[1]. Because we are dealing with collaborative problem solving, not all of these actions can be accomplished by one agent alone. Rather, one agent needs to propose an action (the agent is said to *initiate* the collaborative act), and the other accept it (the other agent *completes* the collaborative act).

There are also explicit communication acts involved in collaborative problem solving. Like all communicative acts, these acts are performed by a single agent, but are only successful if the other agent understands the communication. The main communication acts for problem solving include *describe* (e.g. elaborate on an objective, describe a particular solution), *explain* (e.g. provide a rationale for a solution or decision), and *identify* (e.g. communicate the existence of a resource, select a goal to work on). These communication acts, of course, may be used to accomplish other problem solving goals as well. For instance, one might initiate the creation of an objective by describing it.

2.3 Task Manager

The behaviors of the IM, BA and GM are defined in terms of the abstract problem solving model. The details of what these objects are in a particular domain, and how operations are performed, are specified in the Task Manager (TM). The TM supports operations intended to assist in both the *recognition* of what the user is doing with respect to the task at hand and the *execution* of problem solving steps intended to further progress on the task at hand.

Specifically, the Task Manager must be able to:

1. Answer queries about objects and their role in the task/domain (e.g. is an ambulance a resource? Is loading a truck an in-domain plannable/executable action? Is "evacuating a city" a possible in-domain goal?)

2. Provide the interface between the generic problem solving acts used by the BA (e.g. create a solution) and the actual task-specific agents that perform the tasks (e.g. build a course of action to evacuate the city using two trucks)

3. Provide intention recognition services to the IM (e.g. can "going to Avon" plausibly be an extension of the current course of action?)

In our architecture, the Task Manager maps abstract problem solving acts onto the capabilities of the knowledge-based agents at its disposal. For example, in one of our planning domains, the Task Manager uses a planner, router, scheduler, and temporal knowledge base to answer queries and create or modify plans.

[1]This list is not meant to be exhaustive, although it has been developed based on our experiences building systems in several problem solving domains.

2.4 Interpretation Manager

The Interpretation Manager (IM) interprets incoming parsed utterances and generates updates to the Discourse Context. First, it produces turn-taking information. With a push-to-talk interface this is simple. When the user presses the button they have taken the turn; when they release it they have released the turn. As we move to open-mike, identifying turn-taking behavior will require more sophisticated interpretation. TRIPS uses an incremental chart parser that will assist in this process by broadcasting constituents as they are recognized.

The principal task of the IM, however, is to identify the intended speech act, the collaborative problem solving act that it furthers, and the system's obligations arising from the interaction. For instance, the utterance "The bridge over the Genesee is blocked" would be interpreted in some circumstances as a problem statement, the intention being to initiate replanning. The IM would broadcast a discourse-level obligation to respond to a statement, and announce that the user has initiated the collaborative problem solving act of identifying a problem as a means of initiating replanning (say, to change the route currently planned). In other circumstances, the same utterance might be recognized as the introduction of a new goal (i.e. to reopen the bridge). The rules to construct these interpretations are based on the abstract problem solving model and specific decisions are made by querying the Task Manager. For instance, in the above example, key questions might be "is there an existing plan using the bridge?" (an affirmative answer indicates the replanning interpretation) and "is making the bridge available a reasonable high-level goal in this domain?" (an affirmative answer indicates the introduce-goal interpretation).

2.5 Generation Manager

The Generation Manager (GM), which performs content planning, receives problem solving goals requiring generation from the Behavioral Agent (BA) and discourse obligations from the Discourse Context. The GM's task is to synthesize these input sources and produce plans (sequences of discourse acts) for the system's discourse contributions.

Because the GM operates asynchronously from the IM, it can be continuously planning. For instance, it is informed when the user's turn ends and can plan simple take-turn and keep-turn acts even in the absence of further information from the IM or the BA, using timing information.

In the case of grounding behaviors and some conventional interactions (e.g. greetings), the GM uses simple rules based on adjacency pairs; no reference to the problem solving state is necessary. In other cases, it may need information from the BA in order to satisfy a discourse obligation. It may also receive goals from the Behavioral Agent that it can plan to satisfy even in the absence of a discourse obligation, for instance when something important changes in the world and the BA wants to notify the user.

The GM can also plan more extensive discourse contributions using rhetorical relations expressed as schemas, for instance to explain a fact or proposal or to motivatea proposed action. It has access to the discourse context as well as to sources for task- and domain-level knowledge.

When the GM has constructed a discourse act or set of acts for production, it sends the act(s) and associated content to the Response Planner, which performs surface generation. The RP comprises several subcomponents; some are template-based, some use a TAG-based grammar, and one performs output selection and coordination. It can realize turn-taking, grounding and speech acts in parallel and in real-time, employing different modalities where useful. It can produce incremental output at two levels: it can produce the output for one speech act before others in a plan are realized; and where there is propositional content, it can produce incremental output within the sentence (cf. [10]). If a discourse act is realized and produced successfully, the GM is informed and sends an update to the Discourse Context.

2.6 Behavioral Agent

As described above, the Behavioral Agent (BA) is responsible for the overall problem solving behavior of the system. This behavior is a function of three aspects of the BA's environment: (1) the interpretation of user utterances and actions in terms of problem solving acts, as produced by the Interpretation Manager; (2) the persistent goals and obligations of the system, in terms of furthering the problem solving task; (3) Exogenous events of which the BA becomes aware, perhaps by means of other agents monitoring the state of the world or performing actions on the BA's behalf.

As we noted previously, most dialogue systems (including previous versions of TRIPS) respond primarily to the first of these sources of input, namely the user's utterances. In some systems (including previous versions of TRIPS) there is some notion of the persistent goals and/or obligations of the system. Often this is implicit and "hard-coded" into the rules governing the behavior of the system. In realistic conversational systems, however, these would take on a much more central role. Just as people do, the system must juggle its various needs and obligations and be able to talk about them explicitly.

Finally, we think it is crucial that conversational systems get out into the world. Rather than simply looking up answers in a database or even conducting web queries, a conversational system helping a user with a real-world task is truly an agent embedded in the world. Events occur that are both exogenous (beyond its control) and asynchronous (occurring at unpredictable times). The system must take account of these events and integrate them into the conversation. Indeed in many real-world tasks, this "monitoring" function constitutes a significant part of the system's role.

The Behavioral Agent operates by reacting to incoming events and managing its persistent goals and obligations. In the case of user-initiated problem solving acts, the BA determines whether to be cooperative and how much initiative to take in solving the joint problem. For example, if the user initiates creating a new objective, the system can complete the act by adopting a new problem solving obligation to find a solution. It could, however, take more initiative, get the Task Manager to compute a solution (perhaps a partial or tentative one), and further the problem solving by proposing the solution to the user.

The BA also receives notification about events in the world and chooses whether to communicate them to the user and/or adopt problem solving obligations about them. For example, if the system receives a report of a heart attack victim needing attention, it can choose to simply inform the user of this fact (and let them decide what to do about it). More likely, it can decide that something should be done about the situation, and so adopt the intention to solve the problem (i.e. get the victim to a hospital).

Thus the system's task-level initiative-taking behavior is determined by the BA, based on the relative priorities of its goals and obligations. These problem-solving obligations determine how the system will respond to new events, including interpretations of user input.

2.7 Infrastructure

The architecture described in this paper is built on an extensive infrastructure that we have developed to support effective communication between the various components making up the conversational system. Space precludes an extended discussion of these facilities, but see [1] for further details.

System components communicate using the Knowledge Query and Manipulation Language (KQML [11]), which provides a syntax and high-level semantics for messages exchanged between agents. KQML message traffic is mediated by a Facilitator that sits at the hub of a star topology network of components. While a hub may seem to be a bottleneck, in practice this has not been a problem. On the contrary, the Facilitator provides a variety of services that have proven indispensable to the design and development of the overall system. These include: robust initialization, KQML message validation, naming and lookup services, broadcast facilities, subscription (clients can subscribe in order to receive messages sent by other clients), and advertisement (clients may advertise their capabilities).

The bottom line is that an architecture for conversational systems such as the one we are proposing in this paper would be impractical, if not impossible, without extensive infrastructure support. While these may seem like "just implementation details," in fact the power and flexibility of the TRIPS infrastructure enables us to design the architecture to meet the needs of realistic conversation *and to make it work*.

3. EXAMPLE

An example will help clarify the relationships between the various components of our architecture and the information that flows between them, as well as the necessity for each.

Consider the situation in which the user asks "Where are the ambulances?" First, the speech recognition components notice that the user has started speaking. This is interpreted by the Interpretation Manager as taking the turn, so it indicates that a TAKE-TURN event has occurred:

```
(tell (done (take-turn :who user)))
```

The Generation Manager might use this information to cancel or delay a planned response to a previous utterance. It can also be used to generate various grounding behaviors (e.g. changing a facial expression, if such a capability is supported). When the utterance is completed, the IM interprets the user's having stopped speaking as releasing the turn:

```
(tell (done (release-turn :who user)))
```

At this point, the GM may start planning (or executing) an appropriate response.

The Interpretation Manager also receives a logical form describing the surface structure of this request for information. It performs interpretation in context, interacting with the Task Manager. In this case, it asks the Task Manager if ambulances are considered resources in this domain. With an affirmative response, it interprets this question as initiating the problem solving act of identifying relevant resources. Note that contextual interpretation is critical—the user wants to know where the usable ambulances are, not where all known ambulances might be. The IM then generates:

1. A message to the Discourse Context recording the user's utterance in the discourse history together with its structural analysis from the parser.

2. A message to the Discourse Context that the system now has an obligation to respond to the question:

```
(tell
 (introduce-obligation
  :id OBLIG1
  :who system
  :what (respond-to
        (wh-question
         :id UTT1
         :who user
         :what (at-loc (the-set ?x
                         (type ?x ambulance))
                       (wh-term ?l
                         (type ?l location)))
         :why (initiate PS1)))))
```

   This message includes the system's obligation, a representation of the content of the question, and a connection to the recognized problem solving act (defined in the message described next). The IM does not specify how the obligation to respond to the question should be discharged.

3. A message to the Behavioral Agent that the user has initiated a collaborative problem solving act, namely attempting to identify a resource:

```
(tell
 (done
  (initiate
   :who user
   :what (identify-resource
         :id PS1
         :what (set-of ?x
                 (type ?x ambulance))))))
```

   This message includes the problem solving act recognized by the IM as the user's intention, and a representation of the content of the question.

When the Discourse Context receives notification of the new discourse obligation, this fact is broadcast to any subscribed components, including the Generation Manager. The GM cannot answer the question without getting a response from the Behavioral Agent. So it adopts the goal of answering, and waits for information from the BA. While waiting, it may plan and produce an acknowledgment of the question.

When the Behavioral Agent receives notification that the user has initiated a problem solving act, one of four things can happen depending on the situation. We will consider each one in sequence.

Do the Right Thing It may decide to "do its part" and try to complete (or at least further) the problem solving. In this case, it would communicate with other components to answer the query about the location of the ambulances, and then send the GM a message like:

```
(request
 (identify-resource
  :who system
  :what (and
         (at-loc amb-1 rochester)
         ...)
  :why (complete :who system :what PS1)))
```

The BA expects that this will satisfy its problem solving goal of completing the identify-resources act initiated by the user, although it can't be sure until it hears back from the IM that the user understood the response.

Clarification The BA may try to identify the resource but fail to do so. If a specific problem can be identified as having caused the failure, then it could decide to initiate a clarification to obtain the information needed. For instance, say the dialogue has so far concerned a particular subtask involving a particular type of ambulances. It might be that the BA cannot decide if it should identify just the ambulances of the type for this subtask, or whether the user wants to know where all usable ambulances are. So it might choose to tell the GM to request a clarification. In this case, the BA retains its obligation to perform the identify-resources act.

Failure On the other hand, the BA may simply fail to identify the resources that the user needs. For instance, the agents that it uses to answer may not be responding, or it may be that the question cannot be answered. In this case, it requests the GM to notify the user of failure, and abandons (at least temporarily) its problem solving obligation.

Ignoring the Question Finally, the BA might decide that some other information is more important, and send that information to the GM (e.g. if a report from the world indicates a new and more urgent task for the user and system to respond to). In this case, the BA retains the obligation to work on the pending problem solving action, and will return to it when circumstances permit.

Whatever the situation, the Generation Manager receives some abstract problem solving act to perform. It then needs to reconcile this act with its discourse obligation OBLIG1. Of course, it can satisfy OBLIG1 by answering the question. It can also satisfy OBLIG1 by generating a clarification request, since the clarification request is a satisfactory response to the question. (Note that the obligation to answer the original question is maintained as a problem solving goal, not a discourse obligation). In the case of a failure, OBLIG1 could be satisfied by generating an apology and a description of the reason the request could not be satisfied. If the BA ignores the question, the GM might apologize and add a promise to address the issue later, before producing the unrelated information. The apology would satisfy OBLIG1. For a very urgent message (e.g. a time critical warning), it might generate the warning immediately, leaving the discourse obligation OBLIG1 unsatisfied, at least temporarily.

The GM sends discourse acts with associated content to the Response Planner, which produces prosodically-annotated text for speech synthesis together with multimodal display commands. When these have been successfully (or partially in the case of a user interruption) produced, the GM is informed and notifies the Discourse Context as to which discourse obligations should have been met. It also gives the Discourse Context any expected user obligations that result from the system's utterances.

The Interpretation Manager uses knowledge of these expectations to aid subsequent interpretation. For example, if an answer to the user's question is successfully produced, then the user has an obligation to acknowledge the answer. Upon receiving an acknowledgment (or inferring an implicit acknowledge), the IM notifies the Discourse Context that the obligation to respond to the question has truly been discharged, and might notify the BA that the collaborative "Identify-Resource" act PS1 has been completed.

4. IMPLEMENTATION

The architecture described in this paper arose from a long-term effort in building spoken dialogue systems. Because we have been able to easily port most components from our previous system into the new one, the system itself has a wide range of capabilities that were already present in earlier versions. Specifically, it handles robust, near real-time spontaneous dialogue with untrained users as they solve simple tasks such as trying to find routes on a train map and planning evacuation of personnel from an island (see [1] for an overview of the different domains we have implemented). The system supports cooperative, incremental development of plans with clarifications, corrections, modifications and comparison of different options, using unrestricted, natural language (as long as the user stays focussed on the task at hand). The new architecture extends our capabilities to better handle the incremental nature of interpretation, the fact that interpretation and generation must be interleaved, and the fact that realistic dialogue systems must also be part of a broader outside world that is not static. The new architecture further clarifies the separation between linguistic and discourse knowledge on one hand, and task and domain knowledge on the other.

We demonstrated an initial implementation of our new architecture in August 2000, providing the dialogue capabilities for an emergency relief planning domain which used simulation, scheduling, and planning components built by research groups at other institutions. Current work involves extending the capabilities of individual components (the BA and GM in particular) and porting the system to a more complex emergency-handling domain [17].

5. RELATED WORK

Dialogue systems are now in use in many applications. Due to space constraints, we have selected only some of these for comparison to our work. They cover a range of domains, modalities and dialogue management types:

- Information-seeking systems [2, 5, 8, 9, 13, 15, 16] and planning systems [4, 14, 18];
- Speech systems [13, 14, 15, 16], multi-modal systems [5, 8, 18] and embodied conversational agents [2, 9];
- Systems that use schemas or frames to manage the dialogue [9, 13, 14, 16], ones that use planning [4], ones that use models of rational interaction [15], and ones that use dialogue grammars or finite state models [5, 8, 18].

Most of the systems we looked at use a standard interpretation–dialogue management–generation core, with the architecture being either a pipeline or organized around a message-passing hub with a pipeline-like information flow. Our architecture uses a more fluid processing model, which enables the differences we outline below.

5.1 Separation of domain/task reasoning from discourse reasoning

Since many dialogue systems are information-retrieval systems, there may be fairly little task reasoning to perform. For that reason, although many of these systems have domain models or databases separate from the dialogue manager [5, 8, 9, 13, 15, 16], they do not have separate task models. By contrast, our system is designed to be used in domains such as planning, monitoring, and design, where task-level reasoning is crucial not just for performing the task but also for interpreting the user's utterances. Separation of domain knowledge and task reasoning from discourse reasoning – through the use of our Task Manager, various world models, the abstract problem solving model and the Behavioral Agent – allows us access to this information without compromising portability and flexibility.

CommandTalk [18], because it is a thin layer over a standalone planner-simulator, has little direct involvement in task reasoning. However, the dialogue manager incorporates some domain-dependent task reasoning, e.g. in the discourse states for certain structured form-filling dialogues.

In the work of Cassell et al [2], the response planner performs deliberative task and discourse reasoning to achieve communicative and task-related goals. In our architecture, there is a separation between task- and discourse-level planning, with the Behavioral Agent handling the first type of goal and the Generation Manager the other.

Chu-Carroll and Carberry's CORE [4] is not a complete system, but does have a specification for input to the response planner that presumably would come from a dialogue manager. The input specification allows for domain, problem solving, belief and discourse-level intentions. Our Interpretation and Generation Managers reason over discourse-level intentions; they obtain information about domain, problem solving and belief intentions from other modules.

The CMU Communicator system has a dialogue manager, but uses a set of domain agents to "handle all domain-specific information access and interpretation, with the goal of excluding such computation from the dialogue management component" [14]. However, the dialogue manager uses task- or domain-dependent schemas to determine its behavior.

5.2 Separation of interpretation from response-planning

Almost all the systems we examined combine interpretation with response planning in the dialogue manager. The architecture outlined by Cassell et al [2], however, separates the two. It includes an understanding module (performing the same kinds of processing performed by our Interpretation Manager); a response planner (performing deliberative reasoning); and a reaction module (which performs action coordination and handles reactive behaviors such as turn-taking). We do not have a separate component to process reactive behaviors; we get reactive behaviors because different types of goals take different paths through our system. Cassell et al's "interactional" goals (e.g. turn-taking, grounding) are handled completely by the discourse components of our system (the Interpretation and Generation Managers); the handling of their "propositional" goals may involve domain or task reasoning and therefore will involve our Behavioral Agent and problem-solving modules.

Fujisaki et al [8] divide discourse processing into a user model and a system model. As in other work [4, 15], this is an attempt to model the beliefs and knowledge of the agents participating in the discourse, rather than the discourse itself. However, interpretation must still be completed before response planning begins. Furthermore, the models of user and system are finite-state models; for general conversational agents more flexible models may be necessary.

6. CONCLUSIONS

We have described an architecture for the design and implementation of conversational systems that participate effectively in realistic practical dialogues. We have emphasized the fact that interpretation and generation must be interleaved and the fact that dialogue systems in realistic settings must be part of and respond to a broader "world outside." These considerations have led us to an architecture in which interpretation, generation, and system behavior are functions of autonomous components that exchange information about both the discourse and the task at hand. A clean separation between linguistic and discourse knowledge on the one hand, and task- and domain-specific information on the other hand, both clarifies the roles of the individual components and improves portability to new tasks and domains.

7. ACKNOWLEDGMENTS

This work is supported by ONR grant no. N00014-95-1-1088, DARPA grant no. F30602-98-2-0133, and NSF grant no. IRI-9711009.

8. REFERENCES

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

[2] J. Cassell, T. Bickmore, L. Campbell, K. Chang, H. Vilhjálmsson, and H. Yan. Requirements for an architecture for embodied conversational characters. In D. Thalmann and N. Thalmann, editors, *Proceedings of Computer Animation and Simulation '99*, 1999.

[3] J. Chu-Carroll and M. Brown. Initiative in collaborative interactions – its cues and effects. In *Proceedings of AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, 1997.

[4] J. Chu-Carroll and S. Carberry. Collaborative response generation in planning dialogues. *Computational Linguistics*, 24(3):355–400, 1998.

[5] N. Dahlbäck, A. Flycht-Eriksson, A. Jönsson, and P. Qvarfordt. An architecture for multi-modal natural dialogue systems. In *Proceedings of ESCA Tutorial and Research Workshop (ETRW) on Interactive Dialogue in Multi-Modal Systems*, 1999.

[6] George Ferguson and James F. Allen. TRIPS: An integrated intelligent problem-solving assistant. In *Proceedings of AAAI-98*, pages 567–573, 1998.

[7] George Ferguson, James F. Allen, Brad W. Miller, and Eric K. Ringger. The design and implementation of the TRAINS-96 system: A prototype mixed-initiative planning assistant. TRAINS Technical Note 96-5, Department of Computer Science, University of Rochester, 1996.

[8] H. Fujisaki, H. Kameda, S. Ohno, K. Abe, M. Iijima, M. Suzuki, and Z. Taketa. Principles and design of an intelligent system for information retrieval over the internet with a multimodal dialogue interface. In *Proceedings of Eurospeech'99*, 1999.

[9] J. Gustafson, N. Lindberg, and M. Lundeberg. The August spoken dialogue system. In *Proceedings of Eurospeech'99*, 1999.

[10] A. Kilger and W. Finkler. Incremental generation for real-time applications. Technical Report RR-95-11, Deutsches Forschungzentrum für Künstliche Intelligenz GmbH (DFKI), Saarbrücken, Germany, 1995.

[11] Yannis Labrou and Tim Finin. A proposal for a new KQML specification. Technical Report CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1997.

[12] C. Matheson, M. Poesio, and D. Traum. Modelling grounding and discourse obligations using update rules. In *Proceedings of NAACL-2000*, 2000.

[13] S. Rosset, S. Bennacef, and L. Lamel. Design strategies for spoken dialog systems. In *Proceedings of Eurospeech'99*, 1999.

[14] A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. Creating natural dialogs in the Carnegie Mellon Communicator system. In *Proceedings of Eurospeech'99*, 1999.

[15] M. Sadek, P. Bretier, and F. Panaget. ARTIMIS: Natural dialogue meets rational agency. In *Proceedings of IJCAI-97*, 1997.

[16] S. Seneff and J. Polifroni. Dialogue management in the Mercury flight reservation system. In *Proceedings of ANLP-NAACL2000 Workshop on Conversational Systems*, pages 11–16, 2000.

[17] A. Stent. The Monroe corpus. Technical Report 728, Department of Computer Science, University of Rochester, March 2000.

[18] A. Stent, J. Dowding, J. Gawron, E. Owen Bratt, and R. Moore. The CommandTalk spoken dialogue system. In *Proceedings of ACL-99*, pages 183–190, 1999.

[19] D. Traum and J. Allen. Discourse obligations in dialogue processing. In *Proceedings of ACL-94*, pages 1–8, 1994.