# Synchronization in an Asynchronous Agent-based Architecture for Dialogue Systems

**Nate Blaylock** and **James Allen** and **George Ferguson**
Department of Computer Science
University of Rochester
Rochester, New York 14627
USA
{blaylock,james,ferguson}@cs.rochester.edu

## Abstract

Most dialogue architectures are either pipelined or, if agent-based, are restricted to a pipelined flow-of-information. The TRIPS dialogue architecture is agent-based and asynchronous, with several layers of information flow. We present this architecture and the synchronization issues we encountered in building a truly distributed, agent-based dialogue architecture.

## 1 Introduction

More and more people are building dialogue systems. Architecturally, these systems tend to fall into two camps: those with pipelined architectures (*e.g.,* (Lamel et al., 1998; Nakano et al., 1999)), and those with agent-based architectures (*e.g.,* (Seneff et al., 1999; Stent et al., 1999; Rudnicky et al., 1999)). Agent-based architectures are advantageous because they free up system components to potentially act in a more asynchronous manner. However, in practice, most dialogue systems built on an agent-based architecture pass messages such that they are basically functioning in terms of a pipelined flow-of-information.

Our original implementation of the TRIPS spoken dialogue system (Ferguson and Allen, 1998) was such an agent-based, pipelined flow-of-information system. Recently, however, we made changes to the system (Allen et al., 2001a) which allow it to take advantage of the distributed nature of an agent-based system. Instead of system components passing information in a pipelined manner (interpretation → discourse management → generation), we allow the subsystems of interpretation, behavior (reasoning and acting) and generation to work asynchronously. This makes the TRIPS system truly distributed and agent-based.

The driving forces behind these changes are to provide a framework for incremental and asynchronous language processing, and to allow for a mixed-initiative system at the task level. We describe these motivations briefly here.

**Incremental Language Processing** In a pipelined (or pipelined flow-of-information) system, generation does not occur until after both the interpretation and reasoning processes have completed. This constraint is not present in human-human dialogue as evidenced by the presence of grounding, utterance acknowledgment, and interruptions. Making interpretation, behavior, and generation asynchronous allows, for example, the system to acknowledge a question while it is still working on finding the answer.

**Mixed-initiative Interaction** Although pipelined systems allow the system to take discourse-level initiative (*cf.* (Chu-Caroll and Brown, 1997)), it is difficult to see how they could allow the system to take task-level initiative in a principled way. In most systems, reasoning and action are driven mostly by interpreted input (*i.e.,* they are reactive to the user's utterances). In a mixed-initiative system, the system's response should be determined not only by user input, but also system goals and obligations, as well as exogenous events. For example, a system with an asynchronous behavior subsystem can inform the user of a new, important event, regardless of whether it is tied to the user's last utterance. On the other hand, in the extreme version of pipelined flow-of-control, behavior cannot do *anything* until the user says something, which is the only way to get the pipeline flowing.

The reasons for our changes are described in further detail in (Allen et al., 2001a). In this paper, we focus on the issues we encountered in developing an asynchronous agent-based dialogue system and their respective solutions, which turn out to be highly related to the process of grounding.

We first describe the general TRIPS architecture and information flow and then discuss the various points of synchronization within the system. We then discuss what these issues mean in general for the implementation of an asynchronous agent-based system.

## 2 TRIPS Architecture

As mentioned above, the TRIPS system[1] (Allen et al., 2000; Allen et al., 2001a; Allen et al., 2001b) is built on an agent-based architecture. Unlike many systems, however, the flow of information within TRIPS is not pipelined. The architecture and information flow between components is shown in Figure 1. In TRIPS, information flows between the three general areas of interpretation, behavior, and generation.

Each TRIPS component is implemented as a separate process. Information is shared by passing KQML (Finin et al., 1997) messages through a central hub, the Facilitator, which supports message logging and syntax checking as well as broadcast and selective broadcast between components.

We first discuss the individual system components and their functions. We then describe the flow of information through the system and illustrate it with an example.

[1]Further details of the TRIPS dialogue system can be found at our website: `http://www.cs.rochester.edu/research/cisd/`

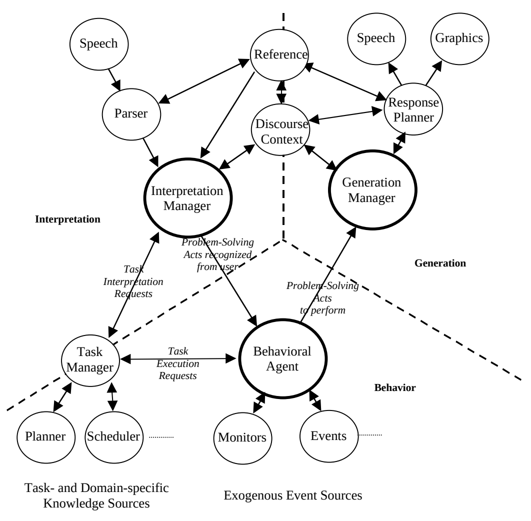

Figure 1: The TRIPS Architecture (Allen et al., 2001a)

### 2.1 System Components

Figure 1 shows the various components in the TRIPS system. Components are divided among three main categories: Interpretation, Behavior, and Generation. As shown in the figure, some components straddle categories, meaning they represent state and provide services necessary for both sorts of processing. The Interpretation Manager (IM) interprets user input coming from the various modality processors as it arises. It interacts with Reference to resolve referring expressions and with the Task Manager (TM) to perform plan and intention recognition, as part of the interpretation process. It broadcasts recognized speech acts and their interpretation as collaborative problem solving actions (see below), and incrementally updates the Discourse Context (DC). The Behavioral Agent (BA) is in some sense the autonomous "heart" of the agent. It plans system behavior based on its own goals and obligations, the user's utterances and actions, and changes in the world state. Actions that require task- and domain-dependent processing are performed by the Task Manager. Actions that involve communication and collaboration with the user are sent to the Generation Manager (GM) in the form of communicative acts. The GM coordinates planning the specific content of utterances and display updates and producing the results. Its behavior is driven by discourse obligations (from the DC), and the directives it receives from the BA.

### 2.1.1 Collaborative Problem Solving Model

The three main components (IM, BA, GM) communicate using messages based on a collaborative problem solving model of dialogue (Allen et al., 2002; Blaylock, 2002). We model dialogue as collaboration between agents which are planning and acting. Together, collaborating agents (*i.e.,* dialogue partners) build and execute plans, deciding on such things as *objectives*, *recipes*, *resources*, *situations* (facts about the world), and so forth. These are called *collaborative problem solving objects*, and are operated on by *collaborative problem solving acts* such as *identity* (present as a possibility), *evaluate*, *adopt*, and others. Thus, together, two agents may decide to *adopt* a certain *objective*, or *identify* a *recipe* to use for an *objective.* The agreed-upon beliefs, objectives, recipes, and so forth constitute the *collaborative problem solving state.*

Of course, because the agents are autonomous, no agent can single-handedly change the collaborative problem solving (CPS) state. *Interaction acts* are actions that a single agent performs to *attempt* to change the CPS state. The interaction acts are *initiate*, *continue*, *complete*, and *reject.* *Initiate* proposes a new change to the CPS state. *Continue* adds new information to the proposal, and *complete* simply accepts the proposal (bringing about the change), without adding additional information. Of course, proposals can be *rejected* at any time, causing them to fail.

As an example, the utterance "Let's save the heart-attack victim in Pittsford" in an emergency planning domain would be interpreted as two interaction acts: `(initiate (identify objective (rescue person1)))` and `(initiate (adopt objective (rescue person1)))`. Here the user is proposing that they consider rescuing person1 as a possible objective to pursue. He is also proposing that they adopt it as an objective to plan for.[2]

*Interaction acts* are recognized (via intention recognition) from speech acts. Interaction acts and speech acts differ in several ways. First, speech acts describe a linguistic level of interaction (*ask*, *tell*, *etc.*), whereas interaction acts deal with a problem solving level (*adopting objectives*, *evaluating recipes* and so forth). Also, as shown above, a single speech act may correspond to many interaction acts.

## 2.2 Information Flow in the System

There are several paths along which information asynchronously flows through the system. We discuss information flow at the levels of problem solving, discourse, and grounding. The section that follows then gives an example of how this proceeds.

### 2.2.1 Problem Solving Level

The problem solving level describes the actual underlying task or purposes of the dialogue and is based on interaction acts. We first describe the problem solving information flow when the user makes an utterance. We then discuss the case where the system takes initiative and how this results in an utterance by the system.

**User Utterance** Following the diagram in Figure 1, when a user makes an utterance, it goes through the Speech Recognizer to the Parser, which then outputs a list of speech acts (which cover the input) to the Interpretation Manager (IM). The IM then sends the speech acts to Reference for resolution.

[2]Here two interaction acts are posited because of the ability of the system to react to each separately, for example *completing* the first, but *rejecting* the second. Consider the possible response "No, not right now." (accept this as a possible objective, but reject adopting it right now), versus "The 911 center in Pittsford is handling that, we don't have to worry about it." (reject this as even a possible objective and reject adopting it). The scope of this paper precludes us from giving more detail about multiple interaction acts.

The IM then sends these speech act hypotheses to the Task Manager (TM), which computes the corresponding interaction acts for each as well as a confidence score that each hypothesis is the correct interpretation.

Based on this, the IM then chooses the best interpretation and broadcasts[3] the chosen CPS act(s) in a "system understood" message. The TM receives this message and updates to the new collaborative problem solving state which this interpretation entails. The Behavioral Agent (BA) receives the broadcast and decides if it wants to form any intentions for action based on the interaction act.

Assuming the BA decides to act on the user's utterance, it sends execution and reasoning requests to the TM, which passes them on to the appropriate back-end components and returns the result to the BA.

The BA then forms an interaction act based on this result and sends it to the GM to be communicated to the user. The GM then generates text and/or graphical updates based on the interaction act and utters/presents them to the user.

In most pipelined and pipelined flow-of-information systems, the only flow of information is at this problem solving level. In TRIPS, however, there are other paths of information flow.

**System Initiative** TRIPS is also capable of taking initiative. As we stated above, this initiative originates in the BA and can come from one of three areas: user utterances, private system objectives, or exogenous events. If the system, say because of an exogenous event, decides to take initiative and communicate with the user, it sends an interaction act to the GM. The GM then, following the same path as above, outputs content to the user.

[3]This is a selective broadcasts to the components which have registered for such messages.

### 2.2.2 Discourse Level

The discourse level[4] describes information which is not directly related to the task at hand, but rather is linguistic in nature. This information is represented as salience information (for Reference) and discourse obligations (Traum and Allen, 1994).

When the user makes an utterance, the input passes (as detailed above) through the Speech Recognizer, to the Parser, and then to the IM, which calls Reference to do resolution. Based on this reference resolved form, the IM computes any discourse obligations which the utterance entails (*e.g.,* if the utterance was a question, to address or answer it, also, to acknowledge that it heard the question).

At this point, the IM broadcasts an "system heard" message, which includes incurred discourse obligations and changes in salience. Upon receipt of this message, Discourse Context updates its discourse obligations and Reference updates its salience information.

The GM learns of new discourse obligations from the Discourse Context and begins to try to fulfill them, regardless of whether or not it has heard from the BA about the problem solving side of things. However, there are some obligations it will be unable to fulfill without knowledge of what is happening at the problem solving level — answering or addressing the question, for example. However, other obligations can be fulfilled without problem solving knowledge — an acknowledgment, for example — in which case, the GM produces content to fulfill the discourse obligation.

If the GM receives interaction acts and discourse obligations simultaneously, it must produce content which fulfills both problem solving and discourse needs. Usually, these interaction acts and discourse obligations are towards the same objective — an obligation to address or answer a question, and an interaction act of identifying a situation (commu-

[4]Although it works in a conceptually similar way, the current system does not handle discourse level information flow quite so cleanly as is presented here. We intend to clean things up and move to this exact model in the near future.

nicating the answer to the user), for example. However, because the system has the ability to take initiative, these interaction acts and discourse obligations may be disparate — an obligation to address or answer a question and an interaction act to identify and adopt a new pressing objective, for example. In this case, the GM must plan content to fulfill the acts and obligations the best it can — apologize for not answering the question and then informing the user, for example. Through this method, the GM maintains dialogue coherence even though the BA is autonomous.

### 2.2.3 Grounding Level

The last level of information flow is at the level that we loosely call grounding (Clark and Schaefer, 1989; Traum, 1994).[5] In TRIPS, acts and obligations are not accomplished and contexts are not updated unless the user has *heard* and/or *understood* the system's utterance.[6]

Upon receiving a new utterance, the IM first determines if it contains evidence of the user having heard and understood the utterance.[7] If the user heard and understood, the IM broadcasts a "user heard" message which contains both salience information from the previous system utterance as well as what discourse obligations the system utterance fulfilled. This message can be used by Reference to update salience information and by Discourse Context to discharge fulfilled discourse obligations.

It is important that these contexts not be updated until the system know that the user heard its last utterance. If the user for example, walks away as the system speaks, the system's discourse obligations will still not fulfilled, and salience information will not change.

The GM receives the "user heard" message and also knows which interaction act(s) the system utterance was presenting. It then broadcasts a "user understood" message, which causes the TM to update the collaborative problem solving state, and the BA to release any goals and intentions fulfilled by the interaction act(s).

Again, it is important that these context updates do not occur until the system has evidence that the user understood its last utterance (for reasons similar to those discussed above).

This handling of grounding frees the system from the assumptions that the user always hears and understands each utterance.

## 2.3 An Example

We use here an example from our TRIPS Medication Advisor domain ((Ferguson et al., 2002)). The Medication Advisor is a project carried out in conjunction with the Center for Future Health at the University of Rochester.[8] The system is designed to help people (especially the elderly) understand and manage their prescription medications.

With the huge growth in the number of pharmaceutical therapies, patients tend to end up taking a combination of several different drugs, each of which has its own characteristics and requirements. For example, each drug needs to be taken at a certain rate: once a day, every four hours, as needed, and so on. Some drugs need to be taken on an empty stomach, others with milk, others before or after meals, and so on. Overwhelmed with this large set of complex interactions many patients simply do not (or cannot) comply with their prescribed drug regimen (Claxton et al., 2001).

The TRIPS Medication Advisor is designed to help alleviate this problem by giving patients easy and accessible prescription information an management in their own home.

For our example, we assume that a dialogue between the system and user is in progress,

[5]TRIPS only uses a small subset of Traum's grounding model. In practice, however, this has not presented problems thus far.

[6]The acceptance or rejection of the actual *content* of an utterance is handled by our collaborative problem solving model (Allen et al., 2002; Blaylock, 2002) and is not further discussed here.

[7]Hearing and understanding are not currently recognized separately in the system. For future work, we would like to extend the system to handle them separately (*e.g.,* the case of the user having heard but not understood).

[8]`http://www.centerforfuturehealth.org`

and a number of other topics have been addressed. At this certain point in the conversation, the system has just uttered "Thanks, I'll try that" and now the user utters the following:

User: "Can I take an aspirin?"

We trace information flow first at the grounding level, then at the discourse level, and finally at the problem solving level. This information flow is illustrated in Figure 2.

**Grounding Level** The utterance goes through the Speech Recognizer and Parser to the IM. As illustrated in Figure 2a, based on the utterance, the IM recognizes that the user heard and understood the system's last utterance, so it sends a "user heard" message, which causes the Discourse Context to update discourse obligations and Reference to update salience based on the system's last utterance.

The GM receives the "user heard" message and sends the corresponding "user understood" message, containing the interaction act(s) motivating the system's last utterance. Upon receiving this message, the TM updates the collaborative problem solving state, and the BA updates its intentions and goals.

Meanwhile ... things have been happening at the discourse level.

**Discourse Level** After the IM sends the "user heard" message, as shown in Figure 2b, it sends Reference a request to resolve references within the user's utterance. It then recognizes that the user has asked a question, which gives the system the discourse obligations of answering (or addressing) the question, as well as acknowledging the question.

The IM then sends a "system heard" message, which causes Reference to update salience and Discourse Context to store the newly-incurred discourse obligations.

The GM receives the new discourse obligations, but has not yet received anything from the BA about problem solving (see below). Without knowledge of what is happening in problem solving, the GM is unable to fulfill the discourse obligation to answer (or address) the question. However, it is able to fulfill the obligation of acknowledging the question, so, after a certain delay of no response from the BA, the GM plans content to produce an acknowledgment, which causes the avatar[9] to graphically show that it is thinking, and also causes the system to utter the following:

System: "Hang on."

Meanwhile ... things have been happening at the problem solving level as well.

**Problem Solving Level** After it sends the "system heard" message, as shown in Figure 2c, the IM computes possible speech acts for the input. In this case, there are two: a yes-no question about the ability to take aspirin and a request to evaluate the action of taking aspirin.

These are sent to the TM for intention recognition. The first case (the yes-no question) does not seem to fit the task model well and receives a low score. (The system prefers interpretations in which the user wants information *for a reason* and not just for the sake of knowing something.) The second speech act is recognized as an initiate of an evaluation of the action of taking aspirin (*i.e.,* the user wants to evaluate this action with the system). This hypothesis receives a higher score.

The IM chooses the second interpretation and broadcasts a "system understood" message that announces this interpretation. The TM receives this message and updates its collaborative problem solving state to reflect that the user did this interaction act. The BA receives the message and, as shown in Figure 2d, decides to adopt the intention of doing the evaluation and reporting it to the user. It sends an evaluation request for the action of the user taking an aspirin to the TM, which queries the back-end components (user knowledge-base and medication knowledge-base) about what prescriptions the user has and if any of them interact with aspirin.

[9]The TRIPS Medication Advisor avatar is a talking capsule whose top half rotates when it is thinking.

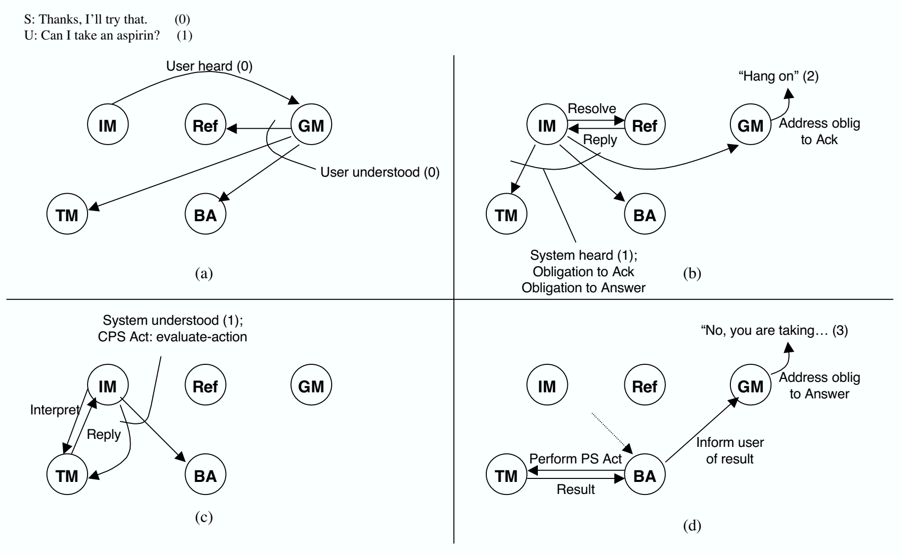

Figure 2: Flow of Information for the Utterance "Can I take an aspirin?" (a) Grounding Level, (b) Discourse Level, (c) and (d) Problem-Solving Level

The back-end components report that the user has a prescription for Celebrex, and that Celebrex interacts with aspirin. The TM then reports to the BA that the action is a bad idea.

The BA then formulates an interaction act reflecting these facts and sends it to the GM. The GM then produces the following utterance, which performs the interaction act as well as fulfills the discourse obligation of responding to the question.

System: "No, you are taking Celebrex and Celebrex interacts with aspirin."

## 3 Synchronization

The architecture above is somewhat idealized in that we have not yet given the details of how the components know which context to interpret messages in and how to ensure that messages get to components in the right order.

We first illustrate these problems by giving a few examples. We then discuss the solution we have implemented.

### 3.1 Examples of Synchronization Problems

One of the problems that faces most distributed systems is that there is no shared state between the agents. The first problem with the architecture described in Section 2 is the lack of context in which to interpret messages. This is well illustrated by the interpret request from the IM to the TM.

As discussed above, the IM sends its candidate speech acts to the TM, which performs intention recognition and assigns a score. The problem is, in which context should the TM interpret utterances? It cannot simply change its collaborative problem solving state each time it performs intention recognition, since it may get multiple requests from the IM, only one of which gets chosen to be the official "interpretation" of the system.

We have stated that the TM updates its context each time it receives a "system understood" or "user understood" message. This brings up, however, the second problem of our distributed system. Because all components are operating asynchronously (including the user, we may add), it is impossible to guarantee that messages will arrive at a component in the desired order. This is because "desired order" is a purely pragmatic assessment. Even with a centralized Facilitator through which all messages must pass, the only guarantee is that messages *from* a particular component *to* a particular component will arrive in order; *i.e.,* if component A sends component B three messages, they will get there in the order that component A sent them. However, if components A and C each send component B a message, we cannot say which will arrive at component B first.

What this means is that the "current" context of the IM may be very different from that of the TM. Consider the case where the system has just made an utterance and the user is responding. As we describe above, the first thing the IM does is check for hearing and understanding and sends off a "user heard" message. The GM, when it receives this message, sends the corresponding "user understood" message, which causes the TM to update to a context containing the system's utterance.

In the meantime, the IM is assuming the context of the systems last utterance, as it does interpretation. It then sends off interpret requests to the TM. Now, if the TM receives an interpret request from the IM *before* it receives the "user understood" message from the GM, it will try to interpret the input in the context of the *user's* last utterance (as if the user had made two utterance in a row, without the system saying anything in between). This situation will give erroneous results and must be avoided.

### 3.2 Synchronization Solution

The solution to these problems is, of course, synchronization: causing components to wait at certain stages to make sure they are in the same context. It is interesting to note that these synchronization points are highly related to a theory of grounding and common ground.

To solve the first problem listed above (lack of context), we have components append context assumptions to the end of each message. Thus, instead of the IM sending the TM a request to interpret $B$, it sends the TM a request to interpret $B$ in the context of having understood $A$. Likewise, instead of the IM requesting that Reference resolve $D$, it requests that Reference resolve $D$ having heard $C$. Having messages explicitly contain context assumptions allows components to interpret messages in the correct context.

With this model, context now becomes discrete, incrementing with every "chunk" of common ground.[10] These common ground updates correspond exactly to the "heard" and "understood" messages we described above. Thus, in order to perform a certain task (reference resolution, intention recognition, *etc.*), a component must know in which common ground context it must be done.

The solution to the second problem (message ordering) follows from explicitly listing context assumptions. If a component receives a message that is appended with a context about which the component hasn't received an update notice (the "heard" or "understood" message), the component simply defers processing of the message until it has received the corresponding update message and can update its context. This ensures that, although messages may not be guaranteed to arrive in the right order, they will be processed in the right context. This provides the necessary synchronization and allows the asynchronous system components to work together in a coherent manner.

## 4 Discussion

We believe that, in general, this has several ramifications for any agent-based, non-pipelined flow-of-information architecture.

1. Agents which are queried about more than one hypothesis must keep state for

[10]For now we treat each utterance as a single "chunk". We are interested, however, in moving to more fine-grained models of dialogue. We believe that our current architecture will still be useful as we move to a finer-grained model.

all hypotheses until one is chosen.

2. Agents cannot assume shared context. Because both the system components and user are acting asynchronously, it is impossible in general for any agent to know what context another agent is currently in.

3. Agents must be able to defer working on input. This feature allows them to wait for synchronization if they receive a message to be interpreted in a context they have not yet reached.

Asynchronous agent-based architectures allow dialogue systems to interact with users in a much richer and more natural way. Unfortunately, the cost of moving to a truly distributed system is the need to deal with synchronization. Fortunately, for dialogue systems, models of grounding provide a suitable and intuitive basis for system synchronization.

## 5 Conclusion and Future Work

In this paper we presented the TRIPS dialogue system architecture: an asynchronous, agent-based architecture, with multiple layers of flow-of-information. We also discussed the problems with building this distributed system. As it turns out, models of grounding provide a foundation for necessary system synchronization.

For future work we plan to "clean up" the model in the ways we have discussed above. We are also interested in moving to a more incremental model of grounding, where grounding can take place and context can change *within* sentence boundaries. Also, we are interested in extending the model to handle asynchronous issues at the turn-taking level. For example, what happens to context when a user barges in while the system is talking, or if the user and system speak simultaneous for a time. We believe we will be able to leverage our asynchronous model to handle these cases.

## 6 Acknowledgments

We would like to thank Amanda Stent, who was involved with the original formulation of this architecture. We also wish to thank the anonymous reviewers for their helpful comments.

This material is based upon work supported by Department of Education (GAANN) grant no. P200A000306; ONR research grant no. N00014-01-1-1015; DARPA research grant no. F30602-98-2-0133; NSF grant no. EIA-0080124; and a grant from the W. M. Keck Foundation.

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 above-mentioned organizations.

## References

J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. 2000. An architecture for a generic dialogue shell. *Journal of Natural Language Engineering special issue on Best Practices in Spoken Language Dialogue Systems Engineering*, 6(3):1–16, December.

James Allen, George Ferguson, and Amanda Stent. 2001a. An architecture for more realistic conversational systems. In *Proceedings of Intelligent User Interfaces 2001 (IUI-01)*, pages 1–8, Santa Fe, NM, January.

James F. Allen, Donna K. Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. 2001b. Towards conversational human-computer interaction. *AI Magazine*, 22(4):27–37.

James Allen, Nate Blaylock, and George Ferguson. 2002. A problem solving model for collaborative agents. In *First International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, July 15-19. To appear.

Nate Blaylock. 2002. Managing communicative intentions in dialogue using a collaborative problem solving model. Technical Report 774, University of Rochester, Department of Computer Science, April.

Jennifer Chu-Caroll and Michael K. Brown. 1997. Initiative in collaborative interactions — its cues and effects. In S. Haller and S. McRoy, editors, *Working Notes of AAAI Spring 1997 Symposium on Computational Models of Mixed Initiative Interaction*, pages 16–22, Stanford, CA.

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

A. J. Claxton, J. Cramer, and C. Pierce. 2001. A systematic review of the associations between dose regimens and medication compliance. *Clinincal Therapeutics*, 23(8):1296–1310, August.

George Ferguson and James F. Allen. 1998. TRIPS: An intelligent integrated intelligent problem-solving assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 567–573, Madison, WI, July.

George Ferguson, James Allen, Nate Blaylock, Donna Byron, Nate Chambers, Myroslava Dzikovska, Lucian Galescu, Xipeng Shen, Robert Swier, and Mary Swift. 2002. The Medication Advisor project: Preliminary report. Technical Report 776, University of Rochester, Department of Computer Science, May.

Tim Finin, Yannis Labrou, and James Mayfield. 1997. KQML as an agent communication language. In J. M. Bradshaw, editor, *Software Agents*. AAAI Press, Menlo Park, CA.

L. Lamel, S. Rosset, J. L. Gauvain, S. Bennacef, M. Garnier-Rizet, and B. Prouts. 1998. The LIMSI ARISE system. In *Proceedings of the 4th IEEE Workshop on Interactive Voice Technology for Telecommunications Applications*, pages 209–214, Torino, Italy, September.

Mikio Nakano, Noboru Miyazaki, Jun ichi Hirasawa, Kohji Dohsaka, and Takeshi Kawabata. 1999. Understanding unsegmented user utterances in real-time spoken dialogue systems. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, pages 200–207.

A. I. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. 1999. Creating natural dialogs in the carnegie mellon communicator system. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech-99)*, pages 1531–1534, Budapest, Hungary, September.

Stephanie Seneff, Raymond Lau, and Joseph Polifroni. 1999. Organization, communication, and control in the Galaxy-II conversational system. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech-99)*, Budapest, Hungary, September.

Amanda Stent, John Dowding, Jean Mark Gawron, Elizabeth Owen Bratt, and Robert Moore. 1999. The CommandTalk spoken dialogue system. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*.

David R. Traum and James F. Allen. 1994. Discourse obligations in dialogue processing. In *Proceedings of the 32nd Annual Meeting of the Association for Computational linguistics (ACL-94)*, pages 1–8, Las Cruces, New Mexico.

David R. Traum. 1994. A computational theory of grounding in natural language conversation. Technical Report 545, University of Rochester, Department of Computer Science, December. PhD Thesis.