Decalog 2007: Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue, pages 91–98.
Trento, Italy, 30 May – 1 June 2007. Edited by Ron Artstein and Laure Vieu.

# Towards Flexible, Domain-Independent Dialogue Management using Collaborative Problem Solving

**Nate Blaylock**
40 South Alcaniz Street
Institute for Human and Machine Cognition (IHMC)
Pensacola, Florida, USA
blaylock@ihmc.us

## Abstract

In this paper, we describe our first efforts at building a domain-independent dialogue manager based on a theory of collaborative problems solving. We describe the implemented dialogue manager and look critically at what level of domain independence was achieved and what remains to be done.

## 1 Introduction

We are interested in building *conversational agents*—autonomous agents which can communicate with humans through natural language dialogue. In order to support dialogue with autonomous agents, we need to be able to model dialogue about the range of activities an agent may engage in, including such things as goal evaluation, goal selection, planning, execution, monitoring, replanning, and so forth.

Current models of dialogue are only able to support a small subset of these sorts of agent activities. Plan-based dialogue models, for example, typically model either planning dialogue (e.g., (Grosz and Sidner, 1990)) or execution dialogue (e.g., (Cohen et al., 1991)), but not both. Also, most plan-based dialogue models make the assumption that agents already have a high-level goal which they are pursuing.

In our previous work (Blaylock and Allen, 2005), we presented a model of dialogue based on collaborative problem solving (CPS), which includes the set of agent activities mentioned above. This CPS-based model of dialogue allows us to model a much wider range of dialogue types and phenomena than previous models.

Besides allowing us to model more complex types of dialogue, it is the hope that CPS dialogue will help with two other important aspects of dialogue: *flexibility* and *portability*. By flexibility, we mean the ability of the system to cover all natural dialogues (i.e., dialogues that humans would naturally engage in) for a given domain. Flexibility is important for naturalness and ease of use, as well as making sure we can understand and incorporate anything the user might say to the system.

Portability refers to to the ease with which the system can be modified to work in new domains. Portability is especially important to the commercial viability of dialogue systems. For dialogue management, our goal is to create a domain-independent dialogue manager that supports "instantiation" to a particular domain through the use of a small amount of domain-specific knowledge. Several recent dialogue managers approach this level of portability ((Larsson, 2002; Bohus and Rudnicky, 2003), inter alia), however, these are based on models of dialogue which do not cover the range of agent activity that we need (see (Blaylock, 2005) for arguments), and they sacrifice some flexibility. Flexibility is lost, as these dialogue managers require a dialogue designer to specify so-called dialogue plans, as part of the domain-specific information fed to the domain-independent dialogue manager. However, these dialogue plans contain not only domain-dependent task knowledge (e.g., the process for making a travel reservation), but also knowledge about how to *interact* with a user about this knowledge (e.g., greet the user, find out travel destination). This essentially puts the onus of dialogue flexibility in the hands of the dialogue system designer, limiting flexibility to the set of dialogues "described" or "encoded" by the dialogue plan. It is our hope that CPS-based dialogue will result in more flexibility and better portability than previous systems

by factoring this interaction knowledge out from domain-dependent task knowledge.

In this paper, we report the progress of our first efforts in building a CPS-based dialogue manager within the SAMMIE-05 dialogue system. We will first briefly describe the CPS dialogue model, and then the SAMMIE-05 dialogue system. We then discuss the implementation SAMMIE-05 dialogue manager and then comment on general progress towards domain independence. We then mention related work and talk about future directions.

## 2 Modeling Dialogue as Collaborative Problem Solving

In this section, we very briefly describe our CPS model of dialogue. Details of the model can be found in (Blaylock and Allen, 2005; Blaylock, 2005). We first describe our model of collaborative problem solving, and then how that is used to model dialogue.

### 2.1 A Model of Collaborative Problem Solving

We see problem solving (PS) as the process by which a (single) agent chooses and pursues *objectives* (i.e., goals). Specifically, we model it as consisting of the following three general phases:

- *Determining Objectives*: In this phase, an agent manages objectives, deciding to which it is committed, which will drive its current behavior, etc.
- *Determining and Instantiating Recipes for Objectives*: In this phase, an agent determines and instantiates a recipe to use to work towards an objective. An agent may either choose a recipe from its recipe library, or it may choose to *create* a new recipe via planning.
- *Executing Recipes and Monitoring Success*: In this phase, an agent executes a recipe and monitors the execution to check for success.

There are several things to note about this general description. First, we do not impose any strict ordering on the phases above. For example, an agent may begin executing a partially-instantiated recipe and do more instantiation later as necessary. An agent may also adopt and pursue an objective in order to help it in deciding what recipe to use for another objective.

It is also important to note that our purpose here is not to specify a specific *problem-solving strategy* or prescriptive model of how an agent *should* perform problem solving. Instead, we want to provide a general descriptive model that enables agents with different PS strategies to still communicate.

Collaborative problem solving (CPS) follows a similar process to single-agent problem solving. Here two agents jointly choose and pursue objectives in the same stages (listed above) as single agents.

There are several things to note here. First, the level of collaboration in the problem solving may vary greatly. In some cases, for example, the collaboration may be primarily in the planning phase, but one agent will actually execute the plan alone. In other cases, the collaboration may be active in all stages, including the planning and execution of a joint plan, where both agents execute actions in a coordinated fashion. Again, we want a model that will cover the range of possible levels of collaboration.

**Examples of Problem-Solving Behavior** In order to better illustrate the problem solving behavior we want to cover in our model, we give several simple examples.

- *Prototypical*: Agent Q decides to go to the park (objective). It decides to take the 10:00 bus (recipe). It goes to the bus stop, gets on the bus and then gets off at the park (execution). It notices that it has accomplished its objective, and stops pursuing it (monitoring).
- *Interleaved Planning and Execution*: Agent Q decides to to go to the park. It decides to take a bus (partial recipe) and starts walking to the bus stop (partial execution) as it decides which bus it should talk (continues to instantiate recipe)....
- *Replanning*: Agent Q decides to go to the park. It decides to walk (objective) and goes outside of the house (begins execution). It notices that it is raining and that doesn't want to walk to the park (monitoring). It decides instead to take the 10:00 bus (replanning)....
- *Abandoning Objective*: Agent Q decides to go to the park by taking the 10:00 bus. As it walks outside, it notices that it is snowing

and decides it doesn't want to go to the park (abandons objective). It decides to watch TV instead (new objective)....

#### 2.1.1 Problem-Solving Objects

The CPS model operates on problem-solving (PS) objects which are represented as typed feature structures. We define an upper-level ontology of such objects, and define the CPS model around them (which helps keep it domain independent). The ontology can then be extended to concrete domains through inheritance and instantiation of the types defined here.

The ontology defines six *abstract PS objects*, from which all other PS objects descend: *objective, recipe, constraint, resource, evaluation,* and *situation.* Types in the model are defines as typed feature structures, and domain knowledge is connected to the ontology by both inheritance in new classes, as well as creating instances of ontological objects.

#### 2.1.2 Collaborative Problem-Solving Acts

We also define a set of actions which operate on these PS objects. Some of these include `identifying` and object for use in problem solving, `adopting` an object for some specific role (e.g., committing to use a particular resource in the plan), `selecting` an objective for execution.

Collaboration cannot be forced by a single-agent, so we define on top of the CPS acts, a model of negotiation, in which agents can negotiate changes to the current CPS state (i.e., the set of PS objects and the agents' joint commitments to them).

### 2.2 Integrating CPS into a Dialogue Model

So far, we have described a model of CPS for any agent-agent collaboration. In order to use CPS to model dialogue, we add an additional layer of grounding based on Traum's grounding model (Traum, 1994), which gives the model coverage of grounding phenomena in language as well.

In modeling dialogue with CPS, we use the CPS state as part of the information state of the dialogue, and the meaning of each utterance (from both parties), can be described as a move in the negotiation of change to the current CPS state (augmented with grounding information). Incidentally, this also allows us to model the intentions of individual utterances in a dialogue.

## 3 The SAMMIE-05 System

The SAMMIE-05 system (Becker et al., 2006)[1] is a multimodal, mixed-initiative dialogue system for controlling an MP3 player. The system can be used to provide typical MP3 services such as playback control, selection of songs/albums/playlists for playback, creation of playlists, and so forth.

The architecture of the SAMMIE-05 system is roughly based on that of the TRIPS system (Allen et al., 2001), in that it separates functionality between subsystems for interpretation, behavior, and generation. Note that this TRIPS-type architecture pushes many tasks typically included in a dialogue manager (e.g., reference resolution) to the interpretation or generation subsystems. The interface in SAMMIE-05 between interpretation, behavior, and generation is, in fact, the CPS-act intentions described in the last section. The intuition behind the TRIPS architecture is to allow a generic behavioral agent to be built, which can drive the dialogue system's behavior by reasoning at a collaborative task level, and not a linguistic level. The dialogue manager we describe in the next section corresponds to what was called the behavioral component in the TRIPS architecture.

## 4 The SAMMIE-05 Dialogue Manager

The SAMMIE-05 dialogue manager supports a subset of the CPS model discussed above. It is implemented as a set of production rules in PATE (Pfleger, 2004). In this section, we report our work towards creating a domain-independent dialogue manager based on our model of collaborative problem solving. It is our hope that the CPS model of dialogue sufficiently abstracts dialogue in such a way that the same set of CPS-based update rules could be used for different domains. We do not yet claim to have a domain-independent CPS-based dialogue manager, although we believe we have made progress towards this end.

Because of the limits of the SAMMIE domain (MP3 player control), many parts of the CPS model have not been encoded into the SAMMIE-05 dialogue manager, and consequently, the dialogue manager cannot be shown to be even a "proof of concept" of the value of the CPS model

[1] Although the SAMMIE system was updated in 2006, in this paper, we describe the SAMMIE system as it existed in December 2005, which we will refer to as the SAMMIE-05 system. It is roughly equivalent to the system described in (Becker et al., 2006).

itself. Our purpose here is, rather, to discuss the progress of CPS-based dialogue management and the insights we gained in encoding a dialogue manager in this (relatively) simple domain.

Important parts of the CPS model which are not supported by the SAMMIE-05 dialogue manager include: collaborative planning and replanning, hierarchical plans, recipe selection, goal abandonment, and most evaluation. Support for these has been left for future work. Phenomena that are covered by the system include: goal selection (albeit not collaborative), collaborative slot-filling, plan execution, and limited evaluation (in the form of feasibility and error checking). As MP3 player control consists of relatively fixed tasks, these phenomena were sufficient to model the kinds of dialogues that SAMMIE-05 handled.

In the rest of this section, we will first describe the dialogue manager, and how we attempt to make it domain independent using abstraction in the PS object hierarchy. In the process of building this dialogue manager, we also discovered some types of domain-specific knowledge *outside* the CPS model proper, which are also necessary for the dialogue manager. This is described as well, and then we describe parts of the dialogue manager which are still domain specific.

### 4.1 High-level Dialogue Management

As with other information state update-based systems, dialogue management in the CPS model can be grouped into three separate processes:

**Integrating Utterance Information** Here the system integrates CPS negotiation acts (augmented with grounding information)—by both user and system—as they are executed. This is a fairly circumscribed process, and is mostly specified in the details of the CPS model itself. This includes such rules as treating an negotiation action as executed when it has been grounded, or marking an object as committed for a certain role when an *adopt* CPS act is successfully executed. These integration rules are detailed in (Blaylock and Allen, 2005).

**Agent-based Control** Once utterance content (and its ensuing higher-level action generation) has been integrated into the dialogue model, the system must decide what to do and what to say next. One of the advantages of the CPS model is that it shields such a process from the linguistic details of the exchange. Instead, we attempt to build such behavior on general collaborative problem-solving principles, regardless of what the communication medium is. We describe this phase in more detail below.

**Package and Output Communicative Intentions** During the first two phases, communicative intentions (i.e., CPS negotiation acts augmented with grounding information) are generated, which the system wants to execute. In this last phase, these communicative intentions are packaged and sent to the generation subsystem for realization. When realization is successfully accomplished, the information state is updated using the rules from the first phase.

The real gain in flexibility and portability from the model comes in the second phase, where the dialogue manager acts more like an autonomous agent in deciding what to do and say next. The information state encodes the agent's commitments (in terms of adopted objectives, etc.), and the current state in the collaborative decision-making process (e.g., which possible objects have been discussed for a certain role). If behavior at this level can be defined on general collaborative problem-solving principles, this would make a precomputed dialogue plan unnecessary. This is a win for both flexibility as well as domain portability.

Most dialogue systems (e.g., GoDiS (Larsson, 2002)) use precomputed *dialogue plans* which define a set of dialogue and domain actions which need to be performed by the system during the dialogue. The need for such an explicit dialogue plan not only adds cost to porting systems, it can also affect the flexibility of the system by restricting the ways it can interact with the user during the dialogue.

### 4.2 Agent-based Control

The agent-based control phase of dialogue management can be divided into three parts. First, the agent tries to fulfill its obligations in the current dialogue (cf. (Traum and Allen, 1994)). This includes principles like attempting to complete any outstanding negotiations on any outstanding CPS acts or at least further them.

Second, the agent looks over its collaborative commitments (as recorded in the CPS state) and attempts to further them. This includes such principles as trying to execute any actions which have been selected for execution. In the case that an objective cannot be executed because vital information is missing (like a value for a parameter), the system will attempt to further the decision making process at that slot (i.e., try to collaboratively find a value for it).

Lastly, the agent uses its own private agenda to determine its actions.[2]

In the SAMMIE-05 dialogue manager, the first and last phases are handled entirely by rules that refer to only the upper-level of the CPS ontology (e.g., *objectives*, *resources*, and so forth), and thus are not dependent on any domain-specific information. Rules in these phases handle the integration semantics of the CPS acts themselves.

The middle level (*agent-based control*) is, however, where portability can become an issue. It is here where the dialogue manager makes decisions about what to do and say next. In our dialogue manager, we were able to formulate many of these rules such that they only access information at the upper ontology level, and do not directly access domain-specific information. As an example, we illustrate a few of these here.

**System Identifies a Resource by Request** The following rule is used identify a resource in response to a request by the user: **if** an identify resource is being negotiated, and the resource has not been stated by the user (i.e., this is a request that the system identify the resource), and the system can uniquely identify such a resource given the constraints used to describe it; **then** add the found resource to the object and create a new continue negotiation of the identify resource CPS act, and add this to the queue of responses to be generated.

As can be seen, this rule relies only on the (abstract) information from the CPS model. In the MP3 domain, this rule is used to provide user-requested sets of information from the database (e.g., in response to "Which Beatles albums do you have?"). No domain-specific knowledge is encoded in this rule.

**System Prepares an Objective to be Executed** The following rule is used when the system marks a top-level objective to be executed next. Note that the current version of the system does not support hierarchical plans, thus the assumption is that this is an atomic action. Also, the system currently assumes that atomic action execution is instantaneous: **if** an objective is in the selected slot (i.e., has been selected for execution) **then** put the objective on a system-internal stack to signal that execution should begin.

This is an example of a simple rule which prepares an *objective* for execution. Similar to the rule just described, no domain-specific information is necessary here—all *objectives* are handled the same, no matter from which domain.

Although we were able to formulate many rules with information available in the CPS model, we encountered some which needed additional information from the domain—including the case where the atomic action execution should actually take place. We now turn our attention to these cases.

### 4.3 Abstracting Additional Domain Information

In the rules discussed above, simple knowledge implicit in the use of abstract PS objects was sufficient for encoding rules. However, there were a few cases which required more information. In this section, we discuss those cases for which we were able to find a solution in order to keep the rules domain-independent. In the next section, we discuss rules which needed to remain domain-specific, and the reasons for that.

Just because domain information is needed for rules does not mean that we cannot write domain-independent rules to handle them. What is required, however, is the specification of an abstraction for this information, which every new domain is then required to provide.

In the MP3 domain, we have identified two general types of this kind of knowledge. We do not consider this to be a closed list:

**Execution Knowledge** One of the example rules above showed how the decision to begin execution of an atomic action is made. However, the

[2]Note that this would prototypically be beliefs, desires and intentions, although the CPS model does not require this. The model itself does not place requirements on single agents and how they are modeled, as long as the agents represent the CPS state and are able to interact using it. The agent we are using for the SAMMIE-05 dialogue manager is not an explicitly represented BDI agent, but rather encodes some simple rules about helpful behavior.

*actual* execution requires knowledge about the domain which is not present in the CPS model (as currently formulated).

In the current system, a domain encodes this information in what we call a *grounded-recipe*, which we have provisionally added as a subtype of *recipe*. A *grounded-recipe* contains a reference to the *objective* it fulfills as well as a pointer to object code (a Java class) which implements an interface for a grounded recipe.

This allows us to write, for example, the following domain-independent rule for atomic action execution in the dialogue manager: **if** an *objective* has been chosen for execution; **then** look up a matching *grounded-recipe* for the *objective* and invoke it (i.e., call the `execute` method of the Java class pointed to in the *grounded-recipe* (passing in the *objective* itself as a parameter)).

**Evaluation of PS Objects** A more general issue which surfaced was the need to make evaluations of various PS objects in order to decide the system's acceptance/rejection of them within a certain context. Although we believe there is a need to specify some sort of general classification for these, only one such evaluation came up in the MP3 domain.

In deciding whether or not to accept the identification of a fully-specified *objective*, the system needed a way of checking the preconditions of the *objective* in order to detect potential errors. For example, the SAMMIE-05 system supports the deletion of a song from a playlist. Now, grounding-level rules (not detailed here) take care of definite reference errors (e.g., mention of a playlist that does not exist). However, if reference to both objects (the song to be deleted and the playlist) is properly resolved, it is still possible, for example, that the user has asked to delete a song from a playlist when that song is not actually on the playlist. Thus, we needed a way of checking this precondition (i.e., does the song exist on the playlist). Similarly, we needed a way of checking to see if the user has requested playback of an empty playlist (i.e., a playlist that does not contain any songs).

As a simple solution, the dialogue manager uses an abstract interface to allow rules to check conditions of any objective: **if** an identify objective is pending for a fully-specified *objective*, and `CheckPreconditions` fails for the *objective*; **then** add a reject of the identify-resource to the output queue.

### 4.4 Domain-specific Rules in the System

Despite our best efforts, a few domain-specific update rules are still present in the dialogue manager. We describe one of these here which was used to cover holes which the CPS model did not adequately address. We hope to expand the model in the future so that this rule can also be generalized.

In the MP3 domain, we support the creation of (regular) playlists as well as so-called auto-playlists (playlists created randomly given constraints). Both of these services correspond to atomic actions in our domain and would be theoretically handled by some of the rules for execution described above. However, these are both actions which actually return a value (i.e., the newly-created playlist). This kind of return value is not currently supported by the CPS model. For this reason, we support the execution of both of these actions with special domain-specific rules.

## 5 Related Work

The work in (Cohen et al., 1991) motivates dialogue as the result of the intentions of rational agents executing joint plans. Whereas their focus was the formal representation of single and joint intentions, we focus on describing and formalizing the interaction itself. We also extend coverage to the entire problem-solving process, including goal selection, planning, and so forth.

Our work is also similar in spirit to work on SharedPlans (Grosz and Sidner, 1990), which describes the necessary intentions for agents to build and hold a joint plan, as well as a high-level sketch of how such joint planning occurs. It defines four operators which describe the planning process: *Select_Rec*, *Elaborate_Individual*, *Select_Rec_GR*, and *Elaborate_Group*. Our CPS acts describe the joint planning process at a more fine-grained level in order to be able to describe contributions of individual utterances. The CPS acts could possibly be seen as a further refinement of the SharedPlans operators. Our model also describes other problem-solving stages, such as joint execution and monitoring.

Collagen (Rich et al., 2001) is a framework for building intelligent interactive systems based on Grosz and Sidner's tripartite model of discourse (Grosz and Sidner, 1986). It provides middleware

for creating agents which act as collaborative partners in executing plans using a shared artifact (e.g., a software application). In this sense, it is similar to the work of Cohen and Levesque described above.

Collagen uses a subset of Sidner's artificial negotiation language (Sidner, 1994) to model individual contributions of utterances to the discourse state. The language defines operators with an outer layer of negotiation (e.g., *ProposeForAccept* (*PFA*) and *AcceptProposal* (*AP*)) which take arguments such as *SHOULD(action)* and *RECIPE*. Our interaction and collaborative problem-solving acts are similar in spirit to Sidner's negotiation language, covering a wider range of phenomena in more detail (including evaluations of goals and recipes, solution constraining, and a layer of grounding).

Perhaps the closest dialogue manager to ours is the TRAINS-93 dialogue manager (Traum, 1996), which was based on some very early notions of collaborative problem solving. Its agentive component, the Discourse Actor, was a reactive controller which acted based on prioritized classes of dialogue states (including discourse obligations, user intentions, grounding, and discourse goals). Our rules were not explicitly prioritized, and, although similar in spirit, the dialogue states in TRAINS-93 were represented quite differently from our CPS model.

## 6 Conclusion and Future Work

We have presented the SAMMIE-05 dialogue manager, which is a first attempt at building a dialogue manager based on collaborative problem solving. Although many parts of collaborative problem solving were not handled by the model, we discussed the extent to which the parts covered were encoded using domain-independent rules based on general principles of collaboration.

There is much future work still to be done. The MP3 player control domain did not exercise large parts of the CPS model, and thus much work remains to be done to fill in the rest of the model. In addition, we have really only scratched the surface in terms of specifying true domain-independent collaborative behavior, including many behaviors which have been detailed in the literature (e.g., (Cohen et al., 1991)). We would like to continue to flesh out this kind of general behavior and add it to the dialogue management rules.

## Acknowledgements

We would like to thank the SAMMIE group at Saarland University and DFKI, especially Jan Schehl, Ciprian Gerstenberger, Ivana Kruijff Korbayová and Tilman Becker, for many helpful discussions in the planning and implementation of this work. This work was funded under the EU-funded TALK project (No. IST-507802).

## References

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

Tilman Becker, Nate Blaylock, Ciprian Gerstenberger, Ivana Kruijff-Korbayová, Andreas Korthauer, Manfred Pinkal, Michael Pitz, Peter Poller, and Jan Schehl. 2006. Natural and intuitive multimodal dialogue for in-car applications: The SAMMIE system. In *Proceedings of the ECAI Sub-Conference on Prestigious Applications of Intelligent Systems (PAIS 2006)*, Riva del Garda, Italy, August 28–September 1.

Nate Blaylock and James Allen. 2005. A collaborative problem-solving model of dialogue. In Laila Dybkjær and Wolfgang Minker, editors, *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pages 200–211, Lisbon, September 2–3.

Nathan J. Blaylock. 2005. Towards tractable agent-based dialogue. Technical Report 880, University of Rochester, Department of Computer Science, August. PhD thesis.

Dan Bohus and Alexander I. Rudnicky. 2003. RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda. In *Proceedings of Eurospeech-2003*, Geneva, Switzerland.

Philip R. Cohen, Hector J. Levesque, José H. T. Nunes, and Sharon L. Oviatt. 1991. Task-oriented dialogue as a consequence of joint activity. In Hozumi Tanaka, editor, *Artificial Intelligence in the Pacific Rim*, pages 203–208. IOS Press, Amsterdam.

Barbara J. Grosz and Candace L. Sidner. 1986. Attention, intention, and the structure of discourse. *Computational Linguistics*, 12(3):175–204.

Barbara J. Grosz and Candace L. Sidner. 1990. Plans for discourse. In P. R. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambridge, MA.

Staffan Larsson. 2002. *Issue-based Dialogue Management*. Ph.D. thesis, Göteborg University.

Norbert Pfleger. 2004. Context based multimodal fusion. In *Sixth International Conference on Multimodal Interfaces (ICMI'04)*, State College, Pennsylvania.

Charles Rich, Candace L. Sidner, and Neal Lesh. 2001. COLLAGEN: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25. Also available as MERL Tech Report TR-2000-38.

Candace L. Sidner. 1994. An artificial discourse language for collaborative negotiation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 814–819, Seattle, WA. Also available as Lotus Technical Report 94-09.

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.

David R. Traum. 1996. Conversational agency: The TRAINS-93 dialogue manager. In *Proceedings of the Twente Workshop on Language Technology 11: Dialogue Management in Natural Language Systems*, pages 1–11, June.