Distributed dialogue management in a blackboard architecture

Antti Kerminen
University of Art and Design Helsinki
{akermine}@uiah.fi

Kristiina Jokinen
University of Helsinki*
{kristiina.jokinen}@helsinki.fi

Abstract

This paper describes a distributed dialogue management scheme for speech-based information seeking dialogue. The dialogue management is distributed to several components, supported by a general blackboard-type architecture for speech systems. By breaking down the dialogue management, we can achieve more general solutions and support sophisticated decision making algorithms.

1 Introduction

The main questions in dialogue system design are: "what should the system do?" and "how should the system do what it is meant to do?" The answer is affected by the task complexity and the desired dialogue functionalities, and also by the architectural design: there is a wide variety of dialogue models that describe processing of the dialogue information on the conceptual level, but there is also a need for generic development platforms that would support experimenting with various architectures and techniques, and would also allow easy extensibility to new functionalities required by new tasks and applications.

As described e.g. in Bernsen et al. (1998), the logical structure of dialogue management contains input analysis (speech recognition and linguistic analysis), dialogue handling, and output generation (natural language generation and speech synthesis). Consequently, spoken dialogue systems have pipelined architectures where the decision making is based on the logical order of the information flow, and dialogue handling is implemented as a single dialogue management component with the task of making decisions of what the system should do next in a particular context.

Three different ways for handling interaction can be distinguished, based on the modelling techniques used for task and dialogue information. The most straightforward way is to use scripts that combine the two types of information under the same plan. Scripts define actions at each dialogue point and can be best used in simple controlling tasks, such as banking and financial services, which require only limited dialogue capabilities that do not go beyond the actual task requirements. Scripts can be written using scripting languages like VoiceXML (VoiceXML Forum, 2003), and including subroutines, it is possible to produce fairly complicated dialogues that contain embedded subdialogues. However, the tight coupling of task and dialogue knowledge does not support natural interaction, and the writing of scripts can be rather time-consuming since all possible action patterns must be explicitly enlisted and described. Also user actions which do not exactly match the responses that the system prompts presuppose, are problematic: elaborate error-handling subroutines are required for each particular case.

A more flexible approach is to separate the domain and dialogue information, and to use forms or frames which define the knowledge necessary to complete the underlying task. The dialogue is thus driven by the information needed, and the interaction appears more natural as the actions can be executed in various orders which are more suitable for individual users (cf. e.g. the question-under-discussion approach implemented in (Lars-

*The research was done while the author was working at the University of Art and Design Helsinki.

son et al., 2001)). For instance, tasks like hotel reservations and flight information providing services contain pieces of knowledge which can be communicated in various orders and combinations between the partners, and thus they do not easily lend themselves to any fixed dialogue modelling. The form also provides a dialogue context in which the actions can be interpreted and planned so as to allow more varied system utterances. Early dialogue research concerned plan-based and BDI-systems (e.g. (Cohen et al., 1990; Jennings, 1993), and references therein), and also emphasized the separation of static task knowledge from the processing knowledge, although the mechanisms for both knowledge representation and reasoning were usually too heavy and complicated for practical dialogue systems.

To enable truly natural interaction, it is necessary to equip the system with knowledge about the task and application as well as with dialogue capabilities. The aim of conversational dialogue management is to improve human-computer interaction by taking human conversational capabilities into account (both verbal and non-verbal communication) and by building models for their computational treatment. It also calls for the development in computational technology, and various statistical and machine-learning techniques have been applied to model conversational phenomena. Research has focussed on modelling such issues as the speakers's intentions (dialogue acts), topic tracking, response planning, cooperation (grounding), and spoken language errors, but more research is still needed to reach a better understanding of the mechanisms behind conversational phenomena, as well as to evaluate which parts of dialogue management will benefit most from the results obtained through the new models. On the other hand, conversational dialogue management can be operationalised by dividing the interaction task into different processing tasks and actions which can be implemented as separate independet procedurs of the whole system, operate asynchroniously, and combine to produce the desired functionality.

Agent-based architectures have thus been introduced with the main benefit being that of flexibility in regard to functioning of the system components (Seneff et al., 1999; Rudnicky et al., 1999; Baylock et al., 2002; Jokinen et al., 2002). While not necessary being committed to conversational dialogue management, agent-based architectures support simultaneous processing of dialogue information by allowing asynchronicity, which provides freedom from a tight pipeline processing and also speeds up the system as e.g. input analysis and output generation can function simultaneously. In practice, however, agent-based architectures usually have general control mechanisms that constrain the components to follow the logical structure of the information flow.

In this paper, we discuss these aspects and present a system with *distributed dialogue management*, where decision making does not manifest itself as a single block, but is distributed over the whole system. We also suggest that this kind of dialogue management provides a useful basis for building prototypes and experimenting with learning techniques presupposed by the more demanding dialogue tasks of conversational dialogue management. In concrete terms, the distribute dialogue management refers to

- interleaving input and output processing with dialogue management
- specifying task management as a separate level of information processing
- introducing special dialogue agents and evaluators as the tools to implement dialogue management.

The paper is organised as follows. Section 2 gives a short overview of the Jaspis framework that the Interact system is built on. Section 3 presents the concept of distributed dialogue management, and its implementation in our system. Section 4 motivates distributed dialogue management from the perspective of learning and adaptivity. Finally, section 5 provides summary and discussion.

2 Jaspis framework

The Interact system is built on top of the Jaspis framework developed by Turunen and Hakulinen (2001). Jaspis is a general blackboard-type architecture for speech-based applications. The two

key features of the architecture are the shared information storage, and the concept of managers, agents and evaluators. The overall architecture, as demostrated in our system, is depicted in Figure 1.

Jaspis can be called an agent-based architecture but the naming of system components can be a little confusing. Especially "agent" in Jaspis is somewhat different from the usual meaning of an autonomous, intelligent agent: it refers to the actions available for a manager. Table 1 gives rough correspondents of Jaspis components and traditional terms of agent architectures.

Jaspis	Agent architecture
manager	agent
evaluator	—
agent	action

Table 1: Correspondence of names in Jaspis and traditional agent architectures.

All information in Jaspis is stored in the Information Storage, which is accessible by all components in the system. Thus, any component can use any information in the system. For example, the dialogue history is accessible by all components and is used e.g. by input agents to analyze the input at the discourse level.

The shared information storage in Jaspis compares to the hub or facilitator of some common agent-based architectures, like the CMU Communicator (Rudnicky et al., 1999), GALAXY-II (Seneff et al., 1999), and TRAINS (Baylock et al., 2002). However, the CMU Communicator, GALAXY-II, and TRAINS allow asynchronous execution of all system components, while in Jaspis the only place for asynchronicity is the Communication Manager, which can handle multiple parallel requests for input/output devices. The other managers cannot run asynchronously but they follow the logical order of information flow. The lack of parallel manager execution makes it difficult to implement grounding or barge-in functionality. On the other hand, within each manager the agents operate independently to the extend of producing scores for how well their functioning fits in a particular system state.

The concept of managers, agents and evaluators is explained in more detail in the following three sections. More information about implementing dialogue management in the Jaspis framework can be found in (Turunen and Hakulinen, 2001).

2.1 Managers

Jaspis has three main managers by default: the Communication Manager, the Dialogue Manager, and the Presentation Manager. The Communication Manager gives a high level abstraction of input/output devices and handles communication between the user and the computer. The Dialogue Manager decides the next system action, and the Presentation Manager realizes the semantic representation of the next system turn as a text and passes it to the output device (speech synthesis, screen). The program flow is controlled by the Interaction Manager, which gives the turn to the managers in a round-robin manner. The managers themselves are responsible for determining if they are capable of doing something in the present situation.

2.2 Agents

As already explained, agents in Jaspis are not agents in the sense that they would be autonomous or intelligent. Rather, they represent the actions available for a manager. Every agent knows how well it is suited for the current situation.

2.3 Evaluators

The evaluators choose the best agent to handle the manager's turn in the program flow. Coordination of the evaluation process is done by the managers. The evaluators evaluate agents by giving a score for each agent. The score can be based on the agent's own opinion, or the evaluator can evaluate agents with its own criteria. Scores from all evaluators are multiplied and this forms the total score for the agent. The agent with a highest score is selected as the action of the manager.

2.4 Devices

Devices are high level abstractions of concrete input/output devices. Below this abstract level there are clients, servers, and engines implementing the actual functionality of the device.

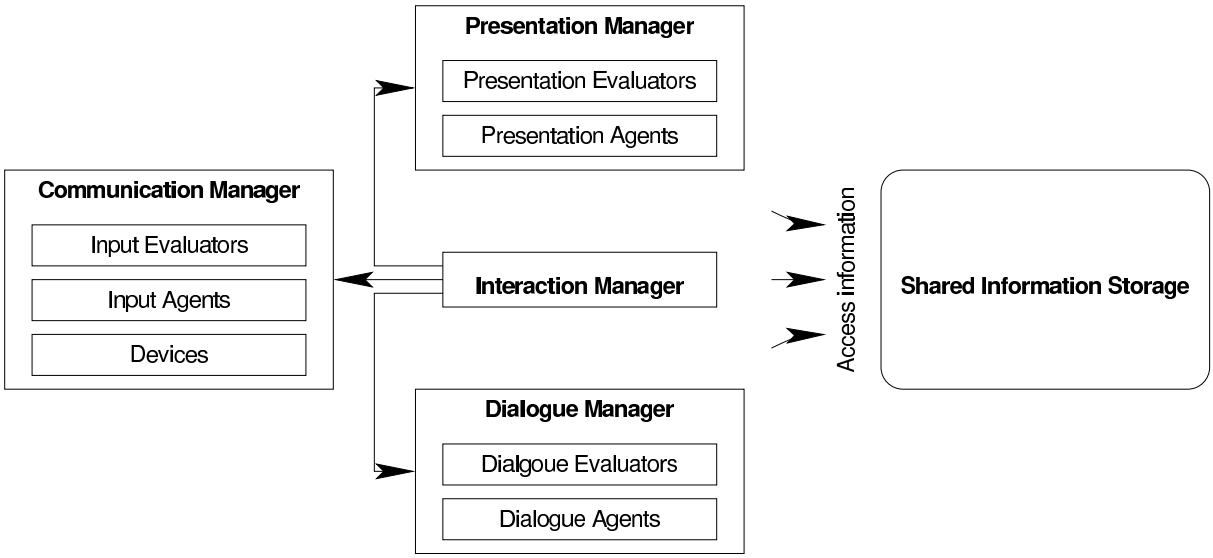

Figure 1: The Jaspis architecture as demonstrated in the Interact system.

An example of a device would be the speech recognizer. The application interacts with an abstract device, and the implementation consists of an client, a server, and an engine for a particular speech recognizer. The Interact system has multimodal input/output capabilities and it also accepts text and touch-screen input and video output.

3 Distributed dialogue management

We have built a timetable service as the Interact demonstration system. The system is capable of answering questions about connections between two places, arrival and departure times, and arrival and departure places. The system supports mixed-initiative dialogues: by default the user has the initiative, but the system enters mixed-initiative mode if the user fails communicate his goals to the system. The general overview of the demostration system is given in (Jokinen et al., 2002).

Unlike the traditional approaches, the dialogue management components are distributed across various managers. The discourse and dialogue level analysis is implemented in the Communication Manager which has a set of input agents and input evaluators to analyze inputs. Input agents in the Communication Manager differ from the agents in the other managers in that they represent different levels of input analysis. Input agents are executed in order, and input evaluators decide when the analysis in complete. The generation of system responses is implemented as presentation agents in the Presentation Manager. The Task Manager takes care of the task related information, while the decision of what to say next is made in the Dialogue Manager.

3.1 Constructive Dialogue Model

Our dialogue model follows the Constructive Dialgue Model (Jokinen, 1996), based on the communicative principles of rational and coordinated interaction (Allwood, 1997; Allwood et al., 2000). The speakers act by exchanging new information and constructing a shared context in which to achieve the underlying goal (in our case this is to provide information from a database, but it can also be a general interactional goal such as "keep the channel open"). The speakers are engaged in a cooperative activity and their actions are constrained by communicative obligations. The emphasis on the principles of ideal co-operation distinguishes our approach from the related work in multiagent systems such as (Gmytrasiewicz and Durfee, 2001) which deals with rationality in decision-theoretic framework, and views communication as a decision process whereby the agent selects acts with the highest expected utility. Although acting may be selected on the basis of cost

and benefit, communication brings in such aspects as trust, obligations and social commitments, the influence of which in dialogue decisions may be difficult to model solely in these terms in practise.

The speakers continue their interaction, and proceed by taking turns to specify, clarify and elaborate the information exchanged as long as the goal is valid, i.e. not yet achieved or abandoned. Each action results in a new dialogue state which is characterized by the agent's perception of the context: the Dialogue Act *Dact*, New Information *NewInfo* and the Topic *Top* of the last utterance, the unfulfilled task goals *TGoals*, expectations, *Expect* are related to communicative obligations, and used to constrain possible interpretations of the next act, and the last speaker. The system's internal states are thus reduced to a combination of these categories, all of which form an independent source of information for the system to decide on the next move (Figure 2). The state description also contains reinforcement values that provide information of the goodness of the state in the given context.

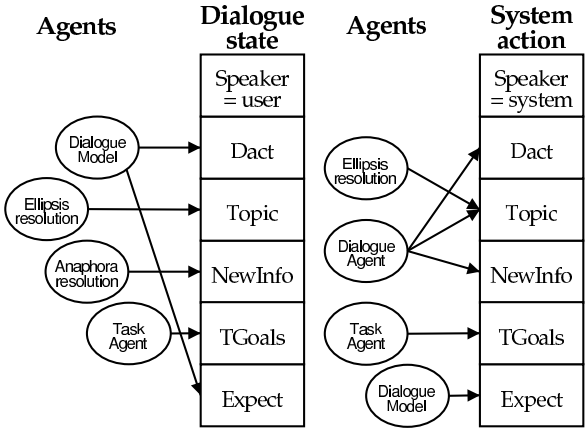

Figure 2: Dialogue states for the user utterance and system action, together with dialogue-level agents involved in producing various information.

3.2 Concepts

The dialogue management operates on a concept-level representation of a user utterance (Figure 2). The conceptual representation includes the dialogue act and the semantic content of the utterance. The semantic content is a set of concepts. Each concept is marked as either Topic or NewInfo, depending on its status with respect to the discourse context. NewInfo concepts encode the information that is asked or presented to the user, while the Topic concepts encode information already available in the context.

3.3 Input analysis

The basic semantic content of the user's utterance is given by the parser. Because the parser only looks at the current utterance, it cannot provide information about ellipses or anaphoric references. However, the discourse-level input agents complete the semantic analysis of the parser by taking into account the dialogue history. There are three input agents in the discourse-level analysis: ellipsis resolution, anaphora resolution, and dialogue act classification. The current system uses a subset of the dialogue acts described in (Jokinen et al., 2001) which presents dialogue act classification based on machine-learning techniques.

In addition there are agents for merging inputs from different input modalities and updating the dialogue history in the information storage.

3.4 Task management

The task management is separate and has its own manager. The tasks of the Task Manager are to make sure that the user has given all required information to complete a database query, execute the complete query, and transfer the information for further processing.

Because of the fairly simple application domain, task specific knowledge is represented as a form with a slot for each concept the system is capable of talking about. The Dialogue Manager and Task Manager communicate with each other via this particular task-form. If the form is filled in so that a database query can be performed, the Task Manager returns the form with all the appropriate parameters filled in, and the Dialogue Manager decides on the Topic/NewInfo status of the concepts corresponding to the parameters and their values with respect to the dialogue situation.

The Task Manager consists of three task agents. The main agent or the DatabaseAgent takes care of the normal access to the database. The two additional agents deal with special situations concern-

ing the execution of a database query. The CursorAgent moves the cursor between pre-fetched database items (e.g. when the user has asked next/previous departure time), and the PlaceAgent ensures that place names in the form are correct and existent in order for a successful query to be executed. The motivation for these two special agents is to keep the main DatabaseAgent simple: by implementing special checks as separate agents, the logic of the different tasks and subtasks becomes clear.

3.5 Dialogue management

In our implementation, dialogue management proper concerns decision making of what to do next, i.e. the selection of the system's next dialogue act. All possible dialogue acts for the system are implemented as dialogue agents, described shortly in the following list:

Open Give the opening prompt.

ReOpen Start the dialogue all over again.

Close Close the dialogue.

Help Give a help message to the user.

Repeat Repeat the previous system's turn.

ReqRepeat Ask the user to repeat the last utterance.

ReqRephrase Ask the user to rephrase the last utterance.

Ask Ask a question from the user.

Inform Give information to the user.

Confirm Ask confirmation from the user.

Note that many — or all — of the agents can be made domain independent. For example, Repeat simply reproduces the system's previous turn from the dialogue history, and requires no application specific information. Also, these kinds of dialogue acts are commonly found in most of the dialogue systems for database access.

The agent selection is done by heuristic rules, implemented so that each agent can tell itself how well it is suited to the current dialogue state. Evaluation is done by a single evaluator, CanHandleEvaluator, which passes all responsibility to agents' own opinion (Figure 3).

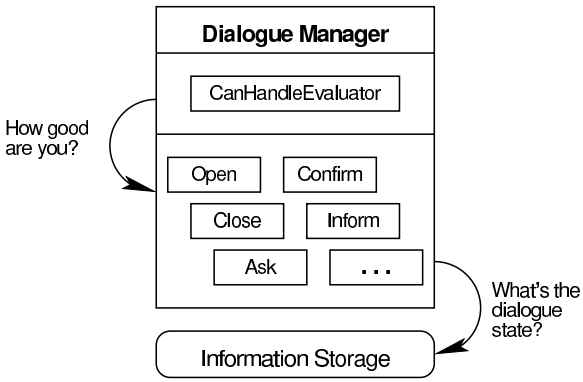

Figure 3: The Dialogue Manager with heuristic agent selection.

The dialogue manager produces the dialgoue act and the semantic content of the system's next turn. The presentation manager is responsible for realizing the intented act as text and passing it to the speech synthesizer.

4 Learning and adaptivity

In this section we outline how the distributed dialogue management scheme supports learning and adaptivity in dialogue systems.

Agent selection by managers in the Jaspis framework compares to action selection by agents in the research area of autonomous agents (Maes, 1994). By defining agents (in Jaspis) as dialogue acts and applying machine learning methods for agent evaluation and selection, it is possible to learn dialogue management strategies and support learning and adaptive dialogue systems at the system-level architecture.

Agent selection with a machine learning algorithm can be implemented with a single evaluator. Now the agents themselves are passive and the decision is made by the evaluator. Figure 4 takes reinforcement learning (RL) as an example. The notion of dialogue states and agents as we have defined them translates directly to the states and actions in reinforcement learning (Sutton and Barto, 1998). The same state representation that was used

in the heuristic decision making by agents can be used by the evaluator that implements the Q-estimate in RL-algorithm, and agents can be seen as actions from which to choose from.

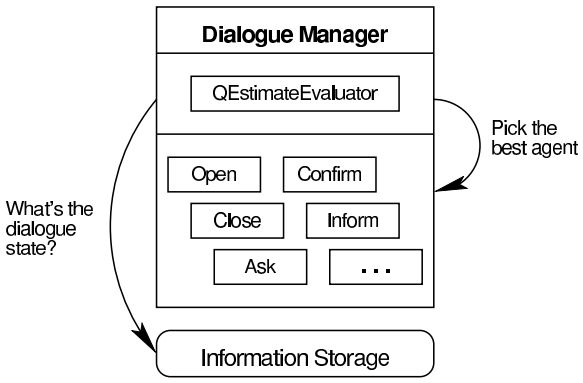

Figure 4: The Dialogue Manager with agent selection based on reinforcement learning.

When the agent selection algorithm is implemented as an evaluator, it can be reused in another application and in a different domain, provided that the dialogue state can be represented in a uniform way. This is indeed often the case: for example Litman et al. (2000) apply RL for the selection of dialogue strategy and use variables such as the (index of the) attribute in focus, the confidence value of the attribute, and the number of times that the attribute value has been asked for, which are generally applicable for different kinds of form-based dialogue systems.

As the Jaspis framework supports agent selection by combining several evaluators, one can take advantage of multiple machine learning methods and design a sophisticated decision making algorithm. This can be done by implementing each machine learning algorithm in an evaluator and combining them with e.g. majority voting.

Reinforcement learning has been used in previous work to learn dialogue strategies (Litman et al., 2000; Walker, 2000; Scheffler and Young, 2002). The choice of RL is natural: training examples for supervised learning methods are hard to get, and the reward given most naturally at end of the dialogue must be propagated back to earlier utterances in the dialogue history.

5 Summary and discussion

We have introduced a distributed dialogue management scheme, and shown how it can support software reuse and the application of machine learning in the selection of dialogue strategy. Building an application partly from existing configurable components allows us to shorten development times and to concentrate on more challenging aspects of dialogue management. On the other hand, the logical structure of dialogue processing also poses the question of task decomposition: what are the smallest parametrised actions that are needed for composing various tasks, and which can thus be used as reusable agents? Further research is needed to provide insight to this question.

One also has to note that a common representation scheme for task and dialogue knowledge is a prerequisite for developing general components for a blackboard architecture. We have adopted the Annotation Graph (AG) framework (Bird and Liberman, 1999) for the basis of our representation of dialogue knowledge. Although the primary target for the AG framework is the annotation of a speech signal with linguistic features by hand, it is a general and flexible model for multiple levels of information attached to a timeline.

The task knowledge in our system is represented with an XML-representation designed especially for the current application. Without any standards for task knowledge representation one can only hope to build a general enough representation that is applicable to various domains. The recent intrest in ontologies and semantic web, as well as building general knowledge bases on the basis of existing classifications, can be seen as a step towards a solution to this kind of portability problems.

The Interact demonstration system that implements distributed dialogue management as described in this paper, will be demonstrated in the workshop. The system can be used with a speech interface, a multimodal interface with a touch-screen, or with a text interface.

Acknowledgements

This research has been carried out in the context of USIX Interact project, a collaboration project between four Finnish universities, and funded by the National Technology Agency, the leading IT companies Fujitsu Invia oyj, Sonera oyj, Lingsoft oy, and Gurusoft oy, as well as the Finnish Assosiation for the Deaf and Arla Institute.

References

J. Allwood, D. Traum, and K. Jokinen. 2000. Cooperation, dialogue and ethics. *International Journal of Human-Computer Studies*, 53(6):871–914.

J. Allwood. 1997. Dialog as collective thinking. In P. Pylkkanen, P. Pylkko, and A. Hautamaki, editors, *Brain, Mind and Physics*. IOS Press, Amsterdam.

N. Baylock, J. Allen, and G. Ferguson. 2002. Synchronization in an asynchronous agent-based architecture for dialogue systems. In K. Jokinen and S. McRoy, editors, *Proceedings of the 3rd SIGDial workshop on Discourse and Dialogue*, pages 1–10, Philadelphia, USA.

N.O. Bernsen, H. Dybkjaer, and L. Dybkjaer. 1998. *Designing Interactive Speech Systems. From First Ideas to User Testing*. Springer Verlag, London.

S. Bird and M. Liberman. 1999. A formal framework for linguistic annotation. Technical Report MS-CIS-99-01, Philadelphia, Pennsylvania.

P.R. Cohen, J. Morgan, and M. Pollack, editors. 1990. *Intentions in Communication*. MIT Press, Cambridge, Massachusettes.

P. Gmytrasiewicz and E. Durfee. 2001. Rational communication in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 4(3):233–272.

N.R. Jennings. 1993. Specifi cation and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *International Journal of Intelligent and Coopeartive Information Systems*, 2(3):289–318.

K. Jokinen, T. Hurtig, K. Hynnä, K. Kanto, A. Kerminen, and M. Kaipainen. 2001. Self-organizing dialogue management. In H. Isahara and Q. Ma, editors, *NLPRS2001 Proceedings of the 2nd Workshop on Natural Language Processing and Neural Networks*, pages 77–84, Tokyo, Japan.

K. Jokinen, A. Kerminen, M. Kaipainen, T. Jauhiainen, G. Wilcock, M. Turunen, J. Hakulinen, J. Kuusisto, and K. Lagus. 2002. Adaptive dialogue systems - interaction with Interact. In K. Jokinen and S. McRoy, editors, *Proceedings of the 3rd SIGDial workshop on Discourse and Dialogue*, pages 64–73, Philadelphia, USA.

K. Jokinen. 1996. Goal formulation based on communicative principles. In *Proceedings of the 16th COLING*, pages 598–603.

S. Larsson, R. Cooper, and S. Ericsson. 2001. menu2dialog. In K. Jokinen, editor, *Proceedings of the 2nd Workshop on Knowledge and Reasoning in Practical Dialogue Systems, IJCAI-2001*, pages 41–45, San Diego.

D. Litman, M. Kearns, S. Singh, and M. Walker. 2000. Automatic optimization of dialogue management. In *Proceedings of COLING 2000*.

P. Maes. 1994. Modeling adaptive autonomous agents. *Artificial Life, I*, (1&2)(9).

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.

K. Scheffer and S. Young. 2002. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proceedings of Human Language Technology*, pages 12–18, San Diego.

S. Seneff, R. Lau, and J. 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)*, pages 1271–1274, Budapest, Hungary.

R. Sutton and A. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusettes.

M. Turunen and J. Hakulinen. 2001. Agent-based adaptive interaction and dialogue management architecture for speech applications. In *Text, Speech and Dialogue. Proceedings of the Fourth International Conference TSD-2001*, pages 357–364.

The VoiceXML Forum. 2003. Voice eXtensible Markup Language VoiceXML, Version 1.00. http://www.voicexml.org.

M. Walker. 2000. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12:387–416.