A Generic Spoken Dialogue Manager Applied to an Interactive 2D Game

Corradini, Andrea; Samuelsson, Christer

Published in:
Lecture Notes in Computer Science

Publication date:
2008

Document version:
Final published version

Citation for pulished version (APA):
Corradini, A., & Samuelsson, C. (2008). A Generic Spoken Dialogue Manager Applied to an Interactive 2D Game. *Lecture Notes in Computer Science*, 2-13.

A Generic Spoken Dialogue Manager Applied to an Interactive 2D Game

Andrea Corradini[1] and Christer Samuelsson[2]

[1] University of Southern Denmark
Institute of Business Communication and Information Science,
6000 Kolding, Denmark
andrea@sitkom.sdu.dk
[2] Umbria, Inc., 1655 Walnut St, Suite 300,
Boulder, CO 80302, USA
christer@umbrialistens.com

Abstract. A generic dialogue manager, previously used in real-world spoken language applications for databases and call-routing, was redeployed in an existing spoken language interface to a 2D game system, which required spatial reasoning, absent in previous applications. This was accomplished by separating general, domain, and application specific knowledge from the machinery, reusing the machinery and the general knowledge, and exploiting ergonomic specification languages for the remaining knowledge. The clear-cut agent-based architecture also contributed strongly to the success of the undertaking.

1 Introduction

Speech and natural language have been increasingly included as modalities to interactive systems. The rationale behind this trend is the assumption that these modes, being natural and common means of communication among humans, would facilitate the user interaction with the machine and simultaneously broaden its bandwidth. While some researchers criticize such an argument based on some evidence that people show different patterns of behavior when they interact with technology as when they interact with each other [Shneiderman, 1980], speech and natural language are of unquestionable benefit in certain domains and for people with special needs and disabilities [Rosenfeld et al, 2001]. Due to current technological limitations, robust and reliable treatment of natural spoken language remains a problem in large domains and even in restricted interactive applications, when recognition errors, misunderstanding and other sources of communication failures must be taken into account. This calls for a system capable to robustly deal with errors and guide users through the interaction process in a collaborative manner.

In state-of-the-art spoken dialogue systems, the dialogue manager is the component in charge of regulating the flow of the conversation between the user and the application. It interprets information gathered from the user and combines it with a number of contextual and internal knowledge sources (such as a dialogue and task history, a domain model and ontology, and a behavioral model of conversational competence)

E. André et al. (Eds.): PIT 2008, LNCS 5078, pp. 2–13, 2008.

in the effort to resolve ambiguities that arise as a consequence of system failures, user mistakes, or under specifications. The dialogue manager eliminates uncertainty through clarification and confirmation requests, either explicitly or implicitly, provides assistance upon request and guides the user by directing the conversation towards a definite goal. In the ideal case, this results in a successful and effective interaction experience, but in practice, there is a trade-off between more user flexibility and better system understanding accuracy. Summarized to a minimum, the task of a dialogue manager consists in reasoning about external observations (such as the user input) in order to update its internal representation of the dialogue and determine what action to perform according to a certain dialogue management policy [Levin et al, 1999].

Several architectures for dialogue management have been proposed for this task. On the one hand, they have many similarities, as they all share a subset of basic core functional components. On the other hand, they exhibit conceptual differences in the way the dialogue state is modeled, and in the specification of the strategy that controls the dialogue flow. Based on these two features (and small variations thereof), architectures for dialogue management can be broadly classified into four main classes. Finite state systems represent dialogue structure in the form of a network, where every node models a question, and the transitions between nodes account for all the possible dialogues [Lemon et al, 2001; McTear, 1998]. The nodes of the network are sometime augmented with scores determined with machine learning techniques, to select optimal network paths [Hurtado et al, 2003]. Finite state systems are theoretically well-understood, and they can be deployed when the interaction constitutes a sequential pre-defined process. Another approach to dialogue management is based on frame structures. Typically, each frame consists of slot-value pairs that are filled in as the interaction proceeds and are also used to guide the user through the dialogue [Hardy et al, 2004; Hochberg et al, 2002; Pieraccini et al, 2001; Seneff and Polifroni, 2000; Ward and Pellom, 1999; Zue et al, 2000]. This strategy is typically deployed in task oriented systems. Its major drawback is scalability. In plan-based architectures [Allen et al, 2001; Bohus and Rudnicky, 2003; Chu-Carroll, 1999; Litman and Allen, 1987; Rudnicky and Xu, 1999] the interaction is driven by a planning process dedicated to achieving certain goals. To accomplish a goal, a sequence of actions (which may be sub-goals themselves) is carried out. The dialogue manager is thus simultaneously a reasoning, inference, and optimization engine, which employs its plans to achieve its goals, given the available information. The merit of such an approach is its ability to handle complex situations, but the drawback is that it quickly becomes computationally intractable. A different recent approach [Bos, 2003; Larsson and Traum, 2000; Lemon et al, 2006] introduces the concept of information state to model dialogues, explaining previous actions, and predicting future actions.

We here deploy an existing, generic, plan-based, slot-filling dialogue manager to an interactive game, where users play a board game using a GUI, as well as spoken and/or typed natural language. The structure of this article is as follows. Section 2 gives a general system overview, while Section 3 focuses on the dialogue manager and provides a worked example. Eventually, a short discussion concludes the paper.

2 System Overview

A sketch of the system architecture is given in Figure 1. It consists of a set of agents that communicate with each other by means of the OAA agent architecture [Cheyer and Martin, 2001].

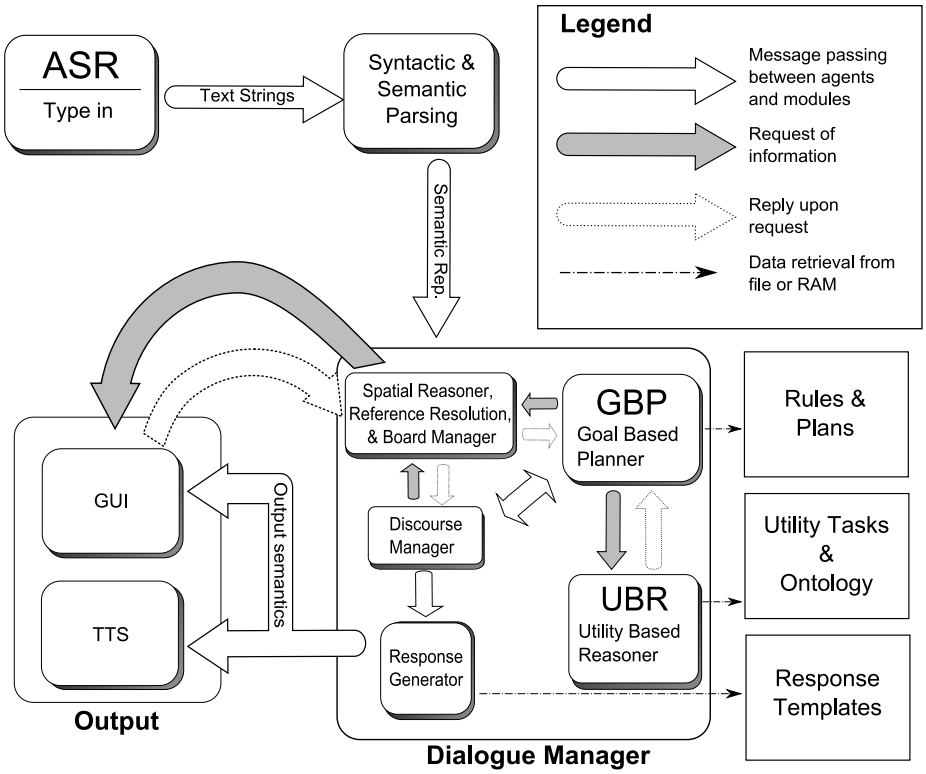

Fig. 1. A sketch of the overall system architecture

2.1 The Pentomino Game Domain

Pentomino is the popular board game of choice for our application. This puzzle game is named after the well-known Domino from which it differs only because its pieces are formed by joining five, instead of just two, equal sized squares together along complete edges. Excluding isomorphism (i.e. rotations, flipping and combinations of them), there are only twelve possible pieces. In Pentomino the player has to use up the set of pieces and land them into a predefined grid-shaped game board (see Figure 2). From a mathematical perspective, Pentomino is a particular type of Polyominos [Golomb, 1965] and as such it is considered an exact mathematical science.

Here, it was realized as a set of agents, controlled by a main GUI manager, which implemented the rules, logic, and presentation of the game, game position, and game history. It allows users to play the game through spoken and/or typed natural language

or direct manipulation, i.e. via a mouse to click, drag, and drop. The user can choose to play in a slightly more challenging way by using the game timer and by attempting at producing a complete solution of the puzzle within the shortest time possible.

2.2 Syntax-Driven Semantic Parser

Players are shown a GUI version of Pentomino (see Figure 2) that they can play using either mouse or keyboard or speech or a combination of two or more of these modalities. In order to allow users play with the game GUI using speech, we use the Sphinx-4 [Walker et al, 2004] open source speech recognizer. Sphinx-4 is a flexible state-of-the-art decoder capable of performing many different types of recognition tasks. It supports all types of HMM-based acoustic models, all standard types of language models, and several search strategies. It can be used along with a JSGF grammar or with an N-gram language model in which case syntactic parsing is not performed by Sphinx-4. Currently, our parser can be used in case where either Sphinx is run using a JSGF[1] grammar or the decoder is bypassed and input is typed-in.

The spoken or typed user input is parsed into a term structure in several steps. The first step is done either by the ASR engine, using a recognition grammar, or by a separate syntactic parser, using the very same grammar. Both of these work as a recursive transition network, transducing the input speech or text into a flat semantic representation, consisting of a sequence of pairs and triples. The semantic parser then chunks this sequence, and each chunk is matched against a set of predicates to create one or more term-structured semantic forms. This output is similar to the predicate-argument structure produced by a syntactic parser using application-specific compositional semantics (see [Corradini et al, 2007] for more details). Such semantic forms constitute the input to the dialogue manager (DM).

2.3 Dialogue Manager Assistants

The dialogue manager itself, and its key components, are described in the next section. It does however have some assistants, from which it can request additional information. Among them, the spatial reasoner, which informs the dialog manager about spatial relations between given Pentomino pieces, and the board manager, which provides the location, shape, and color of all Pentomino pieces, are among these important components. They report to either the GUI module or the DM (in Figure 1 they are actually depicted within the DM module itself), but should be seen as independent contractors.

3 The Dialogue Manager Proper

The core dialogue manager is best viewed as a high-level corporate executive officer. It knows virtually nothing about what it's managing, and must constantly request information from its underlings. This analogy is also apt, in the sense that it employs greedy algorithms, but delegates all actual work. This has the advantages that less

[1] Java.sun.com/products/java-media/speech/forDevelopers/JSG

expensive resources can do most of the work; and that the dialog manager is highly portable, and readily redeployed elsewhere.

Key design features of the dialog manager include that it:

1. partitions knowledge three ways: general, domain-, and application-specific, reusing the first two when possible;
2. separates general dialog behavior from domain-specific behavior, handling the former with a goal-based planner and the latter with a utility-based reasoner;
3. formulates all application and domain knowledge in intuitive, high-level specification languages, which non-programmers can easily master;
4. keeps all data representation and interfaces generic and spotlessly clean, and the tasks of each component clear-cut and limited in scope;
5. recasts extra, specialized reasoning in existing terms, or relegates it to dedicated components;
6. and solicits information not only from the user, but interacts freely with other modules, especially with the client back-end system (typically a relational database or CGI-like transaction system)

This stern design austerity has paid off in terms of reusability, and in ease and speed of application development. In fact, the very same dialog manager, together with several of its components, has been used in spoken dialog front-ends to database query systems, call-routing applications, and transaction systems [Chu-Carroll, 1999; Chu-Carroll and Carpenter, 1999].

3.1 Dialogue Manager Internals

The dialog management machinery consists of:

- a goal-based planner;
- a utility-based reasoner;
- a discourse manager;
- a response generator;
- some agent wrapping.

The discourse manager maintains a summary of the dialogue this far. The response generator is responsible for producing text strings to be synthesized by the TTS and valid game commands for the GUI to carry out. The other relevant components are described below. Other supporting components to them can be added: here, for example, a reference resolution resolver to disambiguate references to objects and entities in the game world. The dialogue manager requires the following application-specific resources:

- a domain or application ontology;
- the set of possible back-end transactions;
- a set of plans and executable primitives;
- a set of response templates;
- a utility function;
- an interface to the other system modules.

The first four of these are created using ergonomic specification languages i.e. a specification language appropriate for the skill set of the developer that minimizes his or her cognitive load.

The first two vary little within a given domain and task type. For instance, building a transaction system for the next bank consists mostly in changing the actual names of low-level entities in the ontology and changing some back-end transactions. Clients however differ widely in how they want the system to behave and in the details of its responses, so both third and fourth item (and managing client expectations) outlined above can easily swell to encompass the lion share of all development effort.

The utility function for our current game system is based on arriving at a single transaction i.e. a complete valid (semantically and pragmatically) game command to send to the back-end system for the GUI to carry out. For a call-routing system, it would instead guide the user towards a unique destination. For a relational database, it could be much more complex, factoring in estimated or actual query response times and answer sizes. Although a fascinating area of inquiry, this falls beyond the scope of the current paper.

Interfacing with other modules is typically quite straightforward as it relies on Prolog-like message passing within the OAA framework. The particular interfaces to the semantic parser, the spatial reasoner, and the GUI manager are discussed below.

3.2 Semantics

A set of alternative hypotheses constitute a semantic form (SF). Each hypothesis consists of a set of slot-value pairs (as a matter of fact, we use quads of slot-operator-value-score rather than slot-value pairs; the score measures certainty; the operator handles local quantifiers, when interfacing with relational databases) interpreted as a conjunction. A hypothesis is inconsistent, if it contains multiple values for any key.

Semantic forms can be combined disjunctively, taking the union, and conjunctively, taking the cross product, of their hypotheses. Each back-end system call is formulated as a hypothesis; their disjunction constitutes a SF defining the set of possible transactions. No transaction may be a subset of another one.

The key goal is to find a unique game move, which is a subset of all consistent hypotheses of a particular SF, namely the resulting SF in Step 3 in the Dialogue Logic below, and where no other move is a subset of any of its consistent hypothesis.

3.3 Dialogue Logic

Here an outline of the deployed dialogue manager's logic:

1. The DM gets a user semantic form from the parser.
2. If it is a non-domain SF, it's handled by the goal-based planner. For example, if the user said "*What was that?*", the DM resubmits its latest utterance to the TTS system, possibly after rephrasing it, whichever the plans for achieving this goal under the prevailing dialogue circumstances call for. Or, if the user typed-in "*quit*", the DM will request an appropriate good-riddance message from the response generator, send it to the TTS component, and plunge into the overall system resource pool.

3. If it is a domain SF, the DM requests the current working SF from the discourse manager, combines the two with the transaction SF from the previous section, and delegates finding the best dialogue move, given the resulting SF, to the utility-based reasoner (unless that SF is compatible with a single transaction, see the previous section, in which case the move consists of the corresponding back-end system call).
4. The latter returns the dialogue move with the highest expected cost-benefit, in this case, either a call to the spatial reasoner (which is free of charge and has a non-negative expected benefit) or the user query minimizing the expectation value of the number of remaining possible back-end transactions. User queries and requests to other agents can be assigned different costs, probabilities of success, and prior distributions over the responses.
5. If the best dialogue move requests information from the spatial reasoner, the latter is invoked, the gathered facts are added to the discourse manager's current working semantic form, and off we go to Step 3.
6. If it is a user query, it is sent to the response generator, together with additional discourse information.
7. The response generator selects its highest ranked matching response template. Catch-all templates guarantee that some response, if awkward, is given; more specific templates create wordings of greater grace and clarity. Since users tend to adapt to their conversational partner [Pearson et al, 2006], all responses must be in vocabulary and covered by the recognition grammar.
8. Off we go to Step 1.

This logic resides in the set of plans, most of which can be recycled (modulo client interference) as they characterize the DM as an interlocutor. There must also be plans for when the current working SF allows no back-end transactions, plans to identify and correct user misconceptions, etc.

This is where the domain ontology comes in handy. If two attributes of an ontology entity conflict throughout the SF, e.g., shape and color, this is a golden opportunity for a system response like "*Sorry, there is no red cross. There is a blue cross and two red pieces: one L and the other W-shaped. Which one do you mean?*" to the user input "*Select the red cross*" given that there is no cross-like pieces of red color in the game.

The underlying discourse mechanism is the same as for other clarification requests.

4 Worked Example

A worked example borrowed from an interaction with a human user may clarify the dialogue manager's modus operandi. Let's consider the Pentomino board in Figure 2 and the following log of user and agent message passing.

During this dialogue fragment, these are the messages that are exchanged among the system components. For the sake of clarity, we omit the irrelevant ones. BM stands for board manager; ASR for speech recognizer (or type-in console) RG for response generator; UBR for utility-based reasoner; and DM INT is the parser semantic form, recast in the DM format.

DM to RG: `greetUser`

RG to DM to TTS: "Welcome to Pentomino!"

DM to RG: `initPrompt`

RG to DM to TTS: "What's your move?"

USER to ASR: "*I wish to select the upper piece*"

PARSER to DM: select(piece(col(upper,no,no), id(no),shape(no))

DM INT: [[pieceAct=select,vert=upper]]

DM to BM: `request(piecesOnboard)`

BM to DM: piecesOnboard([
piece(col(upper,left,red),id(3),shape(x)),
piece(col(no,no,cyan),id(5),shape(z)),
piece(col(lower,right,red),id(7),shape(p)),
piece(col(no,no,purple),id(11),shape(l))])

DM INT: [[color=red,hor=left,id=3,shape=x,vert=upper]
:
[color=purple,hor=no,id=11,shape=l,vert=no]]

DM to BM: `request(piecesRestArea)`

BM to DM: piecesRestArea([
piece(col(no,no,red),id(0),shape(v)),
piece(col(no,no,red),id(1),shape(t)),
piece(col(no,left,red),id(2),shape(w)),
piece(col(no,right,red),id(4),shape(u)),
piece(col(no,no,red),id(6),shape(f)),
piece(col(no,no,red),id(8),shape(i)),
piece(col(upper,no,red),id(9),shape(n)),
piece(col(lower,no,red),id(10),shape(y))])

DM INT: [[color=red,hor=no,id=0,shape=v,vert=no]
:
[color=red,hor=no,id=10,shape=y,vert=lower]]

DM to UBR: [[action=pieceAct,color=red,hor=left,id=3,
pieceAct=select,shape=x,vert=upper],

[action=pieceAct,color=red,hor=no,id=9,
pieceAct=select,shape=n,vert=upper]]

UBR to DM to RG: `wh-question(shape,[n,x])`

RG to DM to TTS: "Is the piece n- or x-shaped?"

GUI: both pieces n and x get highlighted

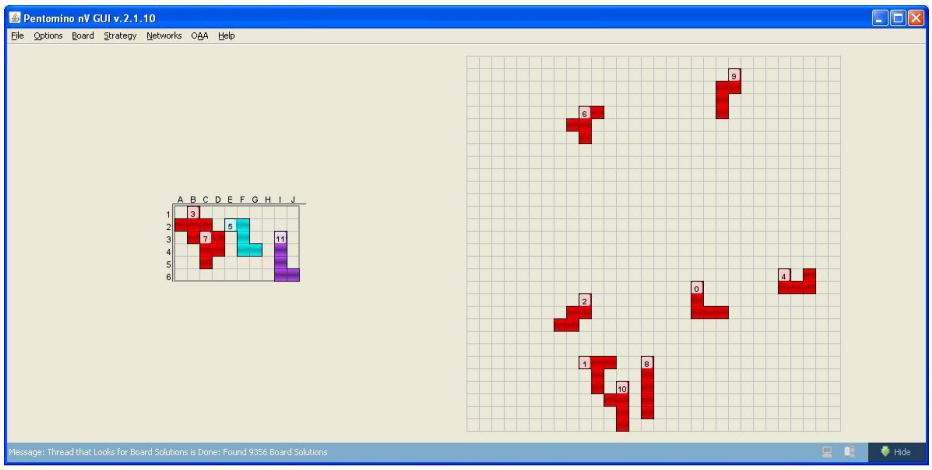

Fig. 2. Graphical interface of the interactive game

Upon quitting the game, the following typical situation occurs in terms of message passing.

DM INT: [[specReq=quit]]

DM to RG: `bidUserFarewell`

RG to DM to TTS: "Thanks for playing Pentomino!" input

The DM is clearly the brain of the system, issuing commands to the other agents. Here, the DM did not need to call on the spatial reasoner, as the semantic parser provided the required spatial information; if some user utterance had instead referred to piece locations relative to other pieces, this would have been necessary.

5 Conclusions

It is said that knowledge is power. Encapsulating knowledge, reusing what can be reused, and providing good specification languages for what remains, allows rapid application development. To stick to our corporate officer metaphor, we doubt that he can be effective without some understanding of what he is managing, but the described DM supports the claim that certain aspects of good management are universal.

There is another good reason for minimizing the DM's knowledge content: changes may go undetected by the DM (in this case, through the use of direct manipulation) and tracking any specific change should be the task of a single component, which, upon request, propagates this change to the rest of the system. Thus, the DM does not assume that the locations of the Pentomino pieces remain the same between turns; it asks the board manager for this info.

Note the three-tier knowledge division: general, domain, and application together with the separation of knowledge from problem solving machinery. The machinery and the general and domain-specific knowledge constitute the reusable resources. Investing in improving these assets yields large dividends.

Our system relies on a clean, standardized interface and agent architecture where each large task is broken down into several limited, clear-cut subtasks. Each component does one thing only, and does it fairly well. We demonstrated that a generic DM, also utilized in real-world spoken language applications for databases and call-routing systems, could be redeployed with little effort in a spoken language interface to a transaction system - in this case, to a board game that requires spatial reasoning, absent in previous applications.

Despite evaluating this kind of DM authoring convenience over multiple different applications is very difficult, preliminary on-going usability studies with human subjects seem to indicate a significant degree of user satisfaction. Dialogue management is still a relatively young science and at this stage of our development, we believe that our empirical experimental results should be weight heavier than any theoretical building. In other words, the DM built and deployed so far has proved its feasibility at all the tasks it has been applied to in our restricted domain.

Currently, we have to expand and elaborate on the templates for clarification requests and system feedback sentences.

Acknowledgments. The EU Marie Curie ToK Programme under grant #FP6-2002-Mobility-3014491 made it possible for the second author to spend a sabbatical period at the University of Potsdam where this research work was started.

References

1. Allen, J., Ferguson, G., Stent, A.: An Architecture for More Realistic Conversational Systems. In: Proc. of the International Conference on Intelligent User Interfaces, pp. 1–8 (2001)
2. Bohus, D., Rudnicky, A.: RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. In: Proceedings of Eurospeech, Geneva, Switzerland, pp. 597–600 (2003)
3. Bos, J., Klein, E., Lemon, O., Oka, T.: DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In: Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue, Sapporo, Japan, pp. 115–124 (2003)
4. Cheyer, A., Martin, D.: The open agent architecture. Autonomous Agents and Multi-Agent System 4(1-2), 143–148 (2001)

5. Chu-Carroll, J.: Form-Based Reasoning for Mixed-Initiative Dialogue Management in Information-Query Systems. In: Proc. of Eurospeech, Budapest, Hungary, pp. 1519–1522 (1999)
6. Chu-Carroll, J., Carpenter, B.: Vector-based natural language call routing'. Computational Linguistics 25(3), 361–388 (1999)
7. Corradini, A., Hanneforth, T., Bak, A.: A Robust Spoken Language Architecture to Control a 2D Game. In: Proc. of AAAI International FLAIRS Conference, pp. 199–204 (2007)
8. Golomb, S.W.: Polyominoes. Scribner's, New York (1965)
9. Hardy, H., Strzalkowski, T., Wu, M., Ursu, C., Webb, N., Biermann, A., Inouye, B., McKenzie, A.: Data-driven strategies for an automated dialogue system. In: Proceedings of the 42nd Meeting of the ACL, Barcelona, Spain, pp. 71–78 (2004)
10. Hochberg, J., Kambhatla, N., Roukos, S.: A flexible framework for developing mixed-initiative dialog systems. In: Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue, Philadelphia, PA, USA, pp. 60–63 (2002)
11. Hurtado, L.F., Griol, D., Sanchis, E., Segarra, E.: A stochastic approach to dialog management. In: Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop, U.S. Virgin Islands, pp. 226–231 (2003)
12. Larsson, S., Traum, D.R.: Information state and dialogue management in the TRINDI dialogue move engine toolkit Source. Natural Language Engineering 6(3-4), 323–340 (2000)
13. Lemon, O., Bracy, A., Gruenstein, A., Peters, S.: The WITAS multi-modal dialogue system I. In: Proceedings of Eurospeech, Aalborg, Denmark, pp. 1559–1562 (2001)
14. Lemon, O., Georgila, K., Henderson, J., Stuttle, M.: An ISU Dialogue System Exhibiting Reinforcement Learning of Dialogue Policies: Generic Slot-filling in the TALK In-car System. In: Proceedings of EACL, Trento, Italy (2006)
15. Levin, E., Pieraccini, R., Eckert, W., di Fabbrizio, G., Narayanan, S.: Spoken language dialogue: From theory to practice. In: Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop, Keystone, CO, USA, pp. 12–15 (1999)
16. Litman, D.J., Allen, J.F.: A plan recognition model for subdialogues in conversations. Cognitive Science 11(2), 163–200 (1987)
17. McTear, M.: Modeling Spoken Dialogues with State Transition Diagrams: Experiences with the CSLU Toolkit. In: Proceedings of International Conference on Spoken Language Processing, Sidney, Australia, pp. 1223–1226 (1998)
18. Pearson, J., Hu, J., Branigan, H.P., Pickering, M.J., Nass, C.I.: Adaptive language behavior in HCI: how expectations and beliefs about a system affect users' word choice. In: Proceedings of the ACM CHI 2006 Conference on Human Factors in Computing Systems, Quebec, Canada, pp. 1177–1180 (2006)
19. Pieraccini, R., Caskey, S., Dayanidhi, K., Carpenter, B., Phillips, M.: ETUDE, A Recursive Dialogue Manager with Embedded User Interface Patterns. In: Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop, Madonna di Campiglio, Italy, pp. 244–247 (2001)
20. Rosenfeld, R., Olsen, D., Rudnicky, A.: Universal speech interfaces. Interactions 8(6), 34–44 (2001)
21. Rudnicky, A., Xu, W.: An Agenda-Based Dialogue Management Architecture for Spoken Language Systems. In: Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop, Keystone, CO, USA, pp. 337–340 (1999)
22. Seneff, S., Polifroni, J.: Dialogue management in the Mercury flight reservation system. In: Proc. of the ANLP-NAACL Workshop on Conversational Systems, pp. 1–6 (2000)

23. Shneiderman, B.: Natural vs. precise concise languages for human operation of computers: research issues and experimental approaches. In: Proceedings of the 18th annual meeting of the Association for Computational Linguistics, pp. 139–141 (1980)
24. Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., Woelfel, J.: Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Sun Microsystems, TR-2004-139 (2004)
25. Ward, W., Pellom, B.: The CU Communicator system. In: Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop, Keystone, CO, USA, pp. 341–344 (1999)
26. Zue, V., Seneff, S., Glass, J., Polifroni, J., Pao, C., Hazen, T.J., Hetherington, L.: Jupiter: A telephone-based conversational interface for weather information. IEEE Transactions on Speech and Audio Processing 8(1), 85–95 (2000)