A Task and Dialogue Model Independent Dialogue Manager

Marcela Charfuelan and **Niels Ole Bernsen**
Natural Interactive Systems Laboratory (NISLab)
University of Southern Denmark
Campusvej 55, DK-5230 Odense M, Denmark
{marcela,nob}@nis.sdu.dk

Abstract

This paper describes the design, development, and early testing of a dialogue manager for a task oriented, multi-domain, multi-lingual, and multi-modal spoken language dialogue system for use in the car. The system was developed in the European project VICO[1]. Main characteristics of the VICO dialogue manager are (i) its task and domain independence, and (ii) a clear separation of task and dialogue model which is reflected in a modular and re-usable architecture.

1 Introduction

This paper presents the design, development, and early testing of a dialogue manager (DM) for a task-oriented, multi-domain, multi-lingual, and multi-modal spoken language dialogue system (SLDS). The DM handles the tasks of address navigation, tourist point of interest navigation, hotel and restaurant reservation, and information about the system itself. The system handles spontaneous input in English, German, and Italian, and outputs speech and display text in those languages.

In recent years, the building of task and domain independent, cooperative, and robust DMs has become increasingly important due to the growing complexity of spoken dialogue applications (Lin *et al.* 99). In the VICO system, for example, a driver who wants to go to a particular destination may also want to select and book a hotel at the destination en route and to make a reservation of a table in the hotel's restaurant. Whilst doing that, the driver might suddenly want to get to the nearest petrol station.

The VICO DM was designed and developed following a rapid prototyping approach. This approach has demonstrated its usefulness for designing and testing different dialogue modelling strategies in a multi-domain framework (Bernsen & Dybkjær 99). Our implementation approach conforms to established software and dialogue engineering techniques (Bernsen *et al.* 98), such as use case modelling and object-oriented software development (Degerstedt & Jönsson 01; O'Neill & McTear 00). An important factor during development was always to keep in mind task and domain independence, which should be reflected in a modular architecture of the DM. The DM should integrate established and new dialogue management techniques whilst keeping the task model description and the dialogue model description as independent as possible (Flycht-Eriksson & Jönsson 00). The main motivation for separating dialogue modelling from task modelling was to create a domain-independent DM architecture in which the main DM engine is able to control an indefinite number of different tasks. Another objective was that the DM architecture should make it easy to define new tasks. Special attention was paid to the representation of the dialogue structure of a task (dialogue modelling) and its particular background knowledge (domain modelling). In what follows, Section 2 presents the VICO DM architecture and the details of dialogue and task modelling. Section 3 explains the multi-domain character of the architecture. Section 4 discusses rapid prototyping and customisation issues. Section 5 presents early evaluation results based on blackbox experiments. Section 6 presents conclusions and describes future work.

2 VICO Dialogue Manager Architecture

The main functionality of a spoken DM can be summarised as follows (Bernsen & Dybkjær 99):

- when required, initiate meta-communication with the user, including repair and clarification meta-communication;
- advance the domain communication, based on a dialogue structure representation of the meaning-in-task-context of the user's input;

[1]Virtual Intelligent Co-Driver: http://www.vico-project.org. The work reported was partially supported by the EC project VICO (IST-2000-25426). We gratefully acknowledge the support.

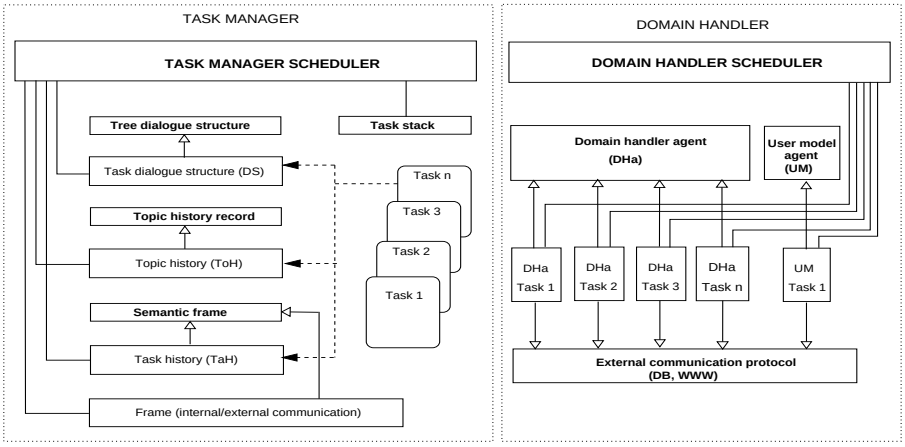

Figure 1: VICO dialogue manager components.

- when appropriate given the user's input, consult the system's domain knowledge base;
- consult the dialogue structure and send the system's response to the language and speech generation components;
- update the context representation (or dialogue history/histories);
- provide support for the speech and language input layers to assist their interpretation of the next user utterance;
- initiate other forms of communication as needed, such as initial greeting and end greeting.

In addition to this classical dialogue management functionality, the VICO DM:

- handles multiple tasks in spontaneous mixed initiative dialogue;
- takes into account combined confidence scores coming from the speech recognition and natural language understanding modules by dynamically modifying the output according to three levels of confidence (high, medium and low).

In McTear's classification of dialogue control techniques (McTear 02), the VICO DM is an event-driven dialogue manager in which dialogue continuation is based on the results of contextual semantic interpretation of the user's utterance and on monitoring changes in the system's belief state. Or, in the classification of dialogue management models in (Xu *et al.* 02), the VICO DM is a DETE (Dialogue model Explicit Task model Explicit) model in which both the dialogue model and the task model are represented explicitly. The dialogue model is represented as a tree or a directed a-cyclic graph, and is dependent on the task. The task model is represented as a finite-state network which is general for the tasks defined in the system and which can easily be extended to describe new tasks. Additional features of the VICO DM:

- a semantic frame is used for *external* communication with the natural language understanding and response generator modules, and *internal* communication with the DM's components, such as dialogue histories, domain agents, etc.
- the DM is divided into two main modules (Figure 1). The task manager is in charge of basic task control using its general system task model, and the domain handler takes care of individual task processing through specialised domain agents.

2.1 Dialogue Modelling

Often, new tasks get added to multi-task systems one by one, so the multi-task DM must be designed for easy addition of dialogue structures for new tasks. Our approach to dialogue modelling for a new task, such as navigation to points of interest (POIs), follows a standard cycle: (1) The task is analysed to determine the complete set of possible task-level contributions from the user. If the task requires a domain database, the user input is (2) translated into a complete set of database queries for the task. (3) All possible database returns are analysed, leading to identification of a set of abstract database return "cases" which must be handled by the dialogue structure. In the POI domain, for instance, one case is the return of more than three hits from the database, such as more than three pharmacies in a city. (4) The main, task-level part of the dialogue structure is designed based on those cases, including the design of output for response generation, triggering confidence score levels, and predictions for the speech recogniser and natural language understanding modules. (5) The dialogue structure is augmented with complete step-by-step dialogue sub-trees for (i) advancing the dialogue to successful completion as well as for (ii) bringing the dialogue back on track through error handling, including, for both (i) and (ii), the design of out-

put for response generation, triggering confidence score levels, and predictions.

The internal representation of the dialogue model is a tree structure. Tree structures have been demonstrated to be powerful tools for various aspects of dialogue modelling. (Ludwig *et al.* 98) use trees to describe the semantic coherence of discourse; (Jönsson 97) uses trees for monitoring the dialogue and recording the focus of the interaction; other examples are (Rudnicky & Xu 99; Lemon *et al.* 02)). In the VICO DM, we use trees to describe the cases into which a task has been subdivided. The nodes represent expected input from the user and include the information necessary for the DM to decide which dialogue move to make next, given reception of expected user input. In section 2.2, we explain the DM's general strategy for handling unexpected user input. The dialogue tree structures are kept in external files that the DM loads during initialisation. The advantage of this approach is twofold: the dialogue structure for a particular task is independent of the current implementation, and debugging and experimentation with the dialogue is reduced to making changes to the tree structure in the external file. Next time the DM loads the dialogue structures, the changes are taken into account.

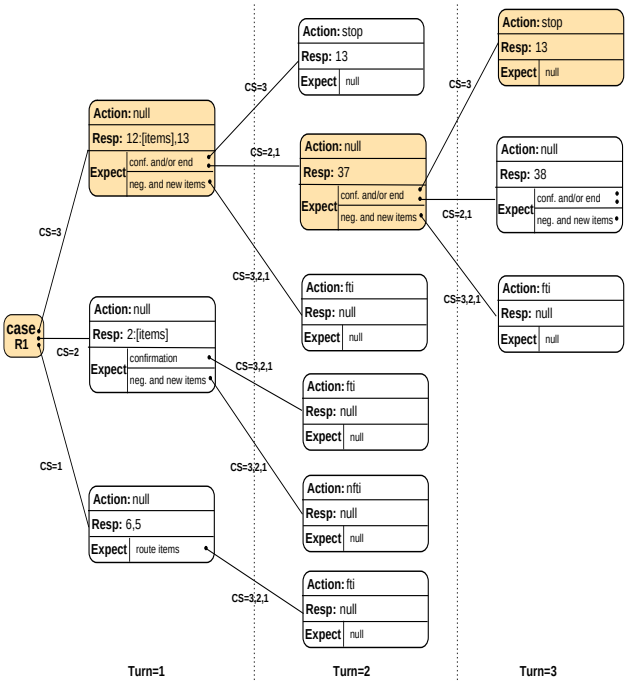

Figure 2: Tree representation of a dialogue model in VICO.

The VICO DM handles three levels of combined confidence score from the speech recogniser and natural language understanding modules. The levels are a main factor determining the system's feedback strategy. When the combined confidence score is high (CS=3), the user gets implicit confirmation of the input; when CS=2 (medium), the user gets explicit confirmation; and when CS= 1 (low), the system initiates a graceful degradation approach. The feedback strategy is represented in the dialogue structure tree as shown in Figure 2. For each input case, a node in the tree is defined for each confidence score. Each node includes information about the response to be generated and/or about the action(s) to be executed in the current state of the dialogue, as well as about the context-dependent expectations to be applied to the next user input. A single node may be used for several confidence scores, such as CS=3,2,1. We use this simplification whenever a particular action, such as re-calculate task status, must be executed no matter the confidence score. For most nodes at which an action is defined, no output response and no expectations are indicated since a new calculation of the dialogue-task status must be performed by the DM first. After action execution we must end up at another node and often in a different task case.

The output responses defined in the dialogue structure are response templates which the DM sends to the response generator in order to produce spoken and display text output. The expectations correspond to types of semantic frame expected for the next user input. Since the dialogue structure for the task being solved (active task) is nearly always consulted after each user turn, the DM is able to send new expectations to the natural language understanding and speech recognition modules, facilitating subsequent input understanding. The expectations are kept turn-by-turn by the DM in a topic history (Section 3.2) and are used to "navigate" through the nodes of the active task tree in order to retrieve the node corresponding to the the current user input.

2.2 Task Modelling

The presence of an explicit system task model, separate from the dialogue models, makes the DM more flexible by making it easier to modify tasks or add new tasks (Flycht-Eriksson 99). The task model in the VICO DM architecture is the finite-state network (FSN) shown in Figure 3. This FSN is a general engine, fully domain-independent and able to handle the interaction flow in a particular

task and among different tasks.

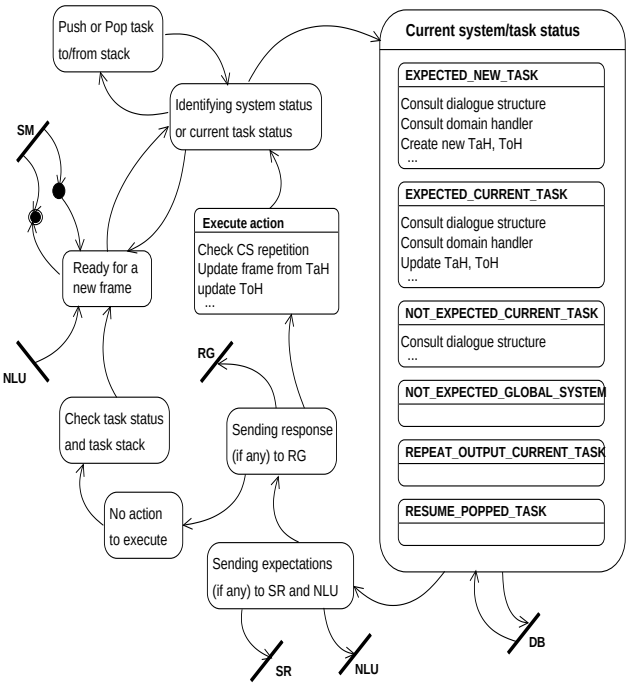

Figure 3: Task model in FSN representation.

For SLDS management, FSNs have been used mainly for describing the dialogue structure and, more recently, for controlling the dialogue (McTear 02). Our system task model is comparable to that presented in (Pieraccini *et al.* 01) in the sense that our FSN has super-states that include embedded FSNs, like the one shown in Figure 3: "Current system/task status". Indeed, this super-state is the core of the task model as it handles the general set of system states. This set includes:

- EXPECTED_NEW_TASK: the input frame is expected and can trigger a new task as no other task is being solved or because in the current active task this type of frame is not expected;
- EXPECTED_CURRENT_TASK: there is a task being solved and the input frame is one of the expected frames for that task;
- NOT_EXPECTED_CURRENT_TASK: a task is being solved (the active task) and the input frame is not among the expected frames for that task;
- NOT_EXPECTED_GLOBAL_SYSTEM: any task is being solved and the input frame does not trigger a new task;
- REPEAT_OUTPUT_CURRENT_TASK: a task is being solved and the user asks for repetition of the last system output;
- RESUME_POPPED_TASK: the user and the system have finished the negotiation of a task and the stack of tasks still has a task pending which is going to be resumed.

We mentioned in Section 4 that the dialogue structure for the active task is consulted almost always after each input frame. This is the case, e.g., the EXPECTED_NEW_TASK state, where a new task is started and the dialogue structure for this task is consulted to determine the response to be sent to the response generator, the expectations to be sent to the natural language understanding and speech recogniser modules, and the action that should be executed, if any.

Depending on whether a task is just starting EXPECTED_NEW_TASK or is active EXPECTED_CURRENT_TASK, the domain handler module may have to be consulted (Figure 1, Section 3). Whenever domain-dependent procedures, facts, or reasoning must be applied, such as determining, for a particular task, if there is enough information to query the external database, the domain handler takes over. The FSN system task model itself only executes a general call procedure to the domain handler module. For a small class of less complex but general cases, the system task model does not consult the domain handler. These cases include simple and general procedures, such as repeating the last system output or generating suitable output when, for whatever reason, including speech recognition error, user not cooperative, etc., the input frame is not expected in the current task nor in the global system.

3 Multi-Domain Architecture

3.1 Task and Domain-Specific Processing

The VICO DM architecture maintains clear separation between domain-independent and domain-dependent procedures. The domain handler module is split into several domain agents, one for each task handled by the system (Figure 1). Only the help task, due to its simplicity and the fact that database querying is not needed, does not have a corresponding domain agent.

The domain handler scheduler receives from the task manager a semantic frame updated with the cumulated knowledge in the task history, and must decide, (i) whether the frame fulfils the requirements for being passed on to a domain agent, and (ii) which domain agent to call. A requirement is, e.g.: do not pass the frame to a domain agent if the confidence score is not highest or the frame contains an ambiguity reported by the natural language understanding module. In these cases, the frame is returned by the domain handler scheduler to the task manager which decides the

next dialogue move based on the current dialogue state and the corresponding dialogue structure. Once a frame is passed to a domain agent, the agent may perform reasoning to decide whether or not to make a database query. Thus, the domain handler scheduler and the domain agents jointly act as filters for evaluating if a database query can be meaningfully made or if the input contains errors which must be resolved prior to database querying. In general, passing the frame to a domain agent and making a database query is only done if the domain handler scheduler/agent are operationally certain of the user's intended input message (high confidence), no errors or inconsistencies have been detected, or a particular input item has been confirmed by the user, e.g. with an explicit "yes" and CS=3. This strategy is intended to avoid growing misunderstandings during dialogue and subsequent, lengthy recovery sub-dialogues.

The domain agents include a generic user model agent for modelling individual drivers based on observations of their input behaviour (Bernsen 03). The driver models are used to adaptively facilitate drivers' accomplishment of complex input tasks. The user modelling module is applied to the task of helping drivers make hotel reservations based on their hotel selection preferences in the past. The hotel reservation agent combines the output of the user modelling module for a particular driver with that driver's initial hotel reservation input, without, of course, overriding any preferences explicitly stated by the driver. The effect is that subsequent hotel database search can proceed on the basis of more constraints than explicitly provided by the driver.

3.2 Multi-task control

To enable the DM handle several tasks as well as embedded tasks, two components support the task manager (Figure 1): (i) the DM uses a set of histories. A task history and a topic history are built for each new task the task manager scheduler creates during dialogue; (ii) the task manager scheduler keeps track of each new task created or completed, using a task stack. The task history collects task-relevant information during interaction. To facilitate user corrections of the knowledge the system is acquiring turn by turn, clear separation is maintained between information provided by the user and information obtained from the domain handler and/or external databases. The topic history records the status of the interaction turn by turn. Our topic history is slightly different from the one in (Bernsen *et al.* 98) because we use the topic history not only for handling repair and clarification meta-communication but for handling interaction in general. In fact, the topic history is used to "navigate" the nodes of the active task tree to retrieve the next response, action, and expectations no matter the status of the task. Consider, e.g., Figure 2 and assume two records in the topic history. The first record shows CS=3 and task case=R1 as determined by the first frame received. The second record shows CS=2, task case still R1, and frame type "conf and/or end" (confirmation and/or end type of frame). The shadowed nodes in Figure 2 constitute the path the task manager goes through to retrieve the appropriate node information after receiving, in the third user input, the frame type "conf. and/or end" and CS=3.

The DM task stack implements, apart from the traditional procedures of push, pop, etc., a logic for handling task interdependencies. When the system is solving, e.g., a navigation task, creation (pushing) of a new navigation task should be avoided until the current one is finished or is popped by the task manager scheduler. Some tasks can be pushed one after the other and popped depending on task completion, as when the user starts a new task when carrying out the ongoing task, e.g. by asking to go to a petrol stations whilst engaged in a hotel reservation task.

4 Rapid Prototyping & Customisation

An advantages of rapid prototyping is to allow us to test and improve ideas from the early stages of development. We implemented, tested, and revised a simple-but-complete early prototype before having specified all the tasks the system should address. Several experiments were made with the early prototype and important information was gained (Bernsen & Dybkjær 01). Examples are that clear separation is needed between task-dependent and task-independent functionality and that the dialogue manager should be completely language-independent.

Figure 4 shows our latest DM prototype with interfaces to other modules. In object-oriented methodology terms, Figure 4 "instantiates" the architecture in Figure 1. In fact, during development of the VICO DM, we have identified

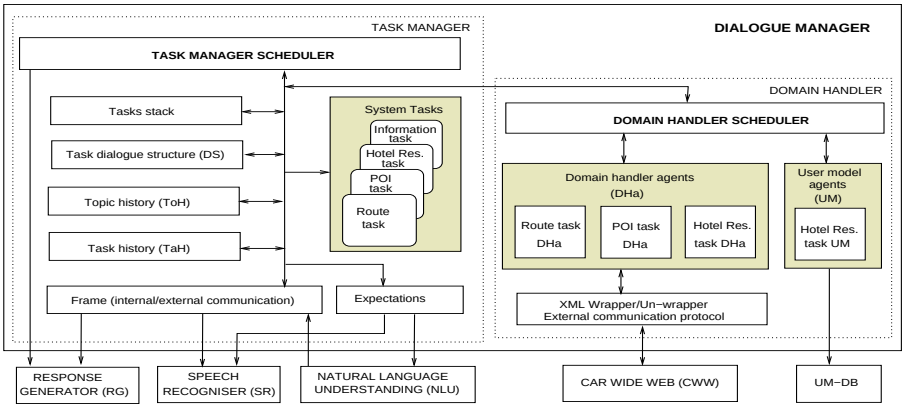

Figure 4: The dialogue manager in the VICO system context.

task-independent procedures for:

- negotiating, handling a task;
- handling of dialogue histories, i.e. task history and topic history;
- representing the dialogue structure in terms of turns, confidence scores, actions, responses and expectations (the tree structure);
- consulting the dialogue structure (general handling of actions, responses, expectations at any point of the dialogue);
- calling or activating domain agents;
- repetition, resumption, greeting procedures.

We have identified task-specific procedures for:

- internal processing in a particular domain agent;
- internal processing in a particular user model.

The steps necessary for customising (or adapting) the current architecture for a new task are:

- define the semantic frame (slots for the new task) and new cases corresponding to new slots;
- define the new task's default expectations (expectations that can trigger the new task);
- define a dialogue model for the new task, cf. Section ;
- implement new actions if necessary;
- implement and add a domain handler if necessary.

We do not claim to have developed the "definitive" object-oriented DM architecture. The above is a first attempt that requires more work, revision and evaluation, but results so far look promising.

5 Dialogue Manager Evaluation

Black box DM evaluation is a complex undertaking. It requires other system components, at least the response generator and data base access, as well as a large and, to the extent possible, systematic set of well-prepared input cases for testing many semantic input variations, given that it is virtually impossible to cover all the input combinatorics. As observed in (Alexandersson & Heisterkamp 00), the DM must not fail on any input, the DM should be robust (no crashes, no loops) before integration in the dialogue system and in particular before the system is tested with real users. We made controlled tests to determine how robust our DM prototype is and which improvements are necessary before user testing. We used an earlier version of our VICO natural language understanding module, our VICO response generator, access to the (partner-provided) database, and no speech recogniser. Diagnostic error analysis, of course, must take into account the errors introduced in the loop by those other modules.

In the first test, we did not simulate speech recognition errors so we assume perfect recognition and confidence score level=3 (highest). The test was intended to test the system's ability to handle as many input cases as possible for navigation to an address (route), navigation to a point of interest (POI), hotel reservation, and restaurant reservation. Table 1 presents the results.

Task	Number of dialogues	Transac. success %	Transac. failure %
Route	31	87.0	12.9
POI	27	66.6	33.3
Hotel res.	19	31.5	68.4
Total	77	66.2	33.7

Table 1: Transaction success testing.

Table 1 shows that the hotel reservation task, in particular, needs revision. Diagnostic analysis showed that most (64.7%) of the errors occured in the hotel domain agent when checking and completing dates. Other errors were introduced by the natural language understanding module.

To test the mixed initiative dialogue and graceful degradation capabilities of the system, the following test was made. We selected a sub-set of user inputs per task and ran these mainly with medium and low confidence scores, primarily us-

Task	Route		POI		Hotel res.	
Confidence score	medium	low	medium	low	medium	low
Average number of turns %	38.2	26.4	33.3	33.3	37.7	28.8
Implicit verification turns %	20.0	14.2	21.4	30.7	0	0
Explicit verification turns %	55.0	0	64.2	0	58.3	38.0
Dialogue degradation turns %	20.0	57.1	28.5	69.2	16.6	71.4
Transaction failure (average)	25.0	50.0	0	25.0	50.0	100.0
Transaction failure (average total)	37.5		12.5		75.0	

Table 2: Mixed initiative and graceful degradation testing.

ing high confidence scores to introduce progress in the dialogue to enable task completion. Some input scenarios included consecutive repetitions of medium and low confidence scores and no high confidence scores in order to force the degradation strategy to finish the dialogue gracefully. Table 2 presents the results.

As expected, transaction failure increases with low confidence score, with a high percentage of dialogue degradation. We still observe a considerable percentage of transaction success for the route and POI tasks with low confidence score. With medium confidence score, the recovery strategy seems to work better. Failure is >25% for route and POI, but too high for the hotel reservation.

6 Conclusions

We have presented a domain and task-independent DM architecture which enables addition of new tasks through "configuring" new dialogue structures and new domain agents. The architecture includes task-dependent dialogue modelling to control domain-dependent procedures, and task-independent system task modelling for general task execution control using a high-level automaton for controlling domain-independent procedures. We found that a high level of DM modularity can be achieved once it has been established which modules or procedures can be generalised and which modules or procedures definitely require task-dependent handling. More work is needed to obtain a robust system. Future work will partly focus on investigating and developing a clean framework for handling task interdependencies like those briefly described in Section 3.2.

References

(Alexandersson & Heisterkamp 00) Jan Alexandersson and Paul Heisterkamp. Some notes on the complexity of dialogues. In *Proceedings of the First Sigdial Workshop on Discourse and Dialogue*, Hong Kong, China, 2000.

(Bernsen & Dybkjær 99) Niels Ole Bernsen and Laila Dybkjær. Draft proposal on best practice methods and procedures in dialogue management. Spoken language dialogue systems and components best practice in development and evaluation (DISC) Deliverable D1.5. http://www.disc2.dk, 1999.

(Bernsen & Dybkjær 01) N. O. Bernsen and Laila Dybkjær. Exploring natural interaction in the car. In *Proceedings of the International Workshop on Information Presentation and Natural Multimodal Dialogue (IPNMD-2001)*, Verona, Italy, 2001.

(Bernsen 03) N. O. Bernsen. User modelling in the car. In *Proceedings of the Ninth International Conference on User Modeling (UM2003)*, Johnstown, USA, 2003. Springer Verlag: Lecture Notes in Artificial Intelligence (to appear).

(Bernsen *et al.* 98) Niels Ole Bernsen, Hans Dybkjær, and Laila Dybkjær. *Designing Interactive Speech Systems: From First Ideas to User Testing.* Springer-Verlag, 1998.

(Degerstedt & Jönsson 01) Lars Degerstedt and Arne Jönsson. A method for iterative implementation of dialogue management. In *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Seatle, USA, 2001.

(Flycht-Eriksson & Jönsson 00) Annika Flycht-Eriksson and Arne Jönsson. Dialogue and domain knowledge management in dialogue systems. In *Proceedings of the First Sigdial Workshop on Discourse and Dialogue*, Hong Kong, China, 2000.

(Flycht-Eriksson 99) Annika Flycht-Eriksson. A survey of knowledge sources in dialogue systems. In *Proceedings of the IJCAI'99 Workshop on Knowledge Reasoning in Practical Dialogue Systems*, Stockholm, Sweden, 1999.

(Jönsson 97) Arne Jönsson. A model for habitable and efficient dialogue management for natural language interaction. *Natural Language Engineering*, 9:103–122, 1997.

(Lemon *et al.* 02) Oliver Lemon, Alexander Gruenstein, Alexis Battle, and Stanley Peters. Multi-tasking and collaborative activities in dialogue systems. In *Proceedings of the Third Sigdial Workshop on Discourse and Dialogue*, Philadelphia, USA, 2002.

(Lin *et al.* 99) Bor-shen Lin, Hsin-min Wang, and Lin-shan Lee. A distributed architecture for cooperative spoken dialogue agents with coherent dialogue state and history. In *Proceedings of the International Workshop on Automatic Speech Recognition and Understanding (ASRU-1999)*, Keystone, Colorado, USA, 1999.

(Ludwig *et al.* 98) Bernd Ludwig, Günther Görz, and Heinrich Niemann. User models, dialog structure, and intentions in spoken dialog. In *Proceedings of KONVENS 98 (Computer, Linguistik und Phonetik zwischen Sprache und Sprechen)*, Bonn, Germany, 1998.

(McTear 02) Michael F. McTear. Spoken dialogue technology: enabling the conversational interface. *ACM Computing Surveys*, 34:90–169, 2002.

(O'Neill & McTear 00) Ian M. O'Neill and Michael F. McTear. Object-oriented modelling of spoken language dialogue systems. *Natural Language Engineering*, 6:341–362, 2000.

(Pieraccini *et al.* 01) Roberto Pieraccini, Sasha Caskey, Krishna Dayanidhi, Bob Carpenter, and Michael Phillips. Etude, a recursive dialogue manager with embedded user interface patterns. In *Proceedings of the International Workshop on Automatic Speech Recognition and Understanding (ASRU-2001)*, Trento, Italy, 2001.

(Rudnicky & Xu 99) A. Rudnicky and W. Xu. An agenda-based dialog management architecture for spoken language systems. In *IEEE Automatic Speech Recognition and Understanding Workshop*, Keystone, Colorado, USA, 1999.

(Xu *et al.* 02) Weiqun Xu, Bo Xu, Taiyi Huang, and Hairong Xin. Bridging the gap between dialogue management and dialogue models. In *Proceedings of the Third Sigdial Workshop on Discourse and Dialogue*, Philadelphia, USA, 2002.