Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

Trung H. Bui

CTIT Dissertation Series No. 08-122
Center for Telematics and Information Technology (CTIT)
P.O. Box 217, 7500AE, Enschede, The Netherlands
ISSN:1381-3617

The research reported in this dissertation has been carried out in the ICIS (Interactive Collaborative Information Systems) project. ICIS is one of nine ICT projects that were selected for funding by the BSIK program of the Dutch government in the fall 2003.

SIKS Dissertation series No. 2008-32
The research reported in this dissertation has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN: 978-90-365-2714-9

Cover design: Trung H. Bui
Printed by PrintPartners Ipskamp, Enschede

TOWARD AFFECTIVE DIALOGUE MANAGEMENT USING PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W. H. M. Zijm,
on account of the decision of the graduation committee
to be publicly defended
on Thursday, October 9, 2008 at 16:45

by

Bui Huu Trung

born on July 5, 1975
in Nghe An, Vietnam

This dissertation is approved by the promotor,
prof. dr. ir. A. Nijholt,
and the assistant-promotor,
dr. J. Zwiers.

In memory of my father Bùi Hữu Độ(1941-2002)

Acknowledgements

First of all, I want to thank my promotor Anton Nijholt for giving me the opportunity to conduct my PhD research in the Human Media Interaction (HMI) group. I received tremendous support from him not only in my research activities but also in solving personal issues during my first year in Twente. Anton also gave me a lot of freedom to choose my research topic and to combine it with my previous research experience.

Next, I would like to thank my daily supervisor Job Zwiers for all his help and advice during my PhD study and especially in helping me to make a good balance on the time spent for the project and my PhD work. I am thankful to Mannes Poel for his guidance as my second daily supervisor. Mannes's comments are always concise and very insightful, especially on the mathematical background presented in this thesis.

The first part of this thesis (Chapters 3 & 4) originates from the work I did in Switzerland under supervision of Martin Rajman. I would like to thank Martin for teaching me how to do research and introducing me to the Natural Language Processing topic. I found his specific advice and direct guidance very helpful during the first period of my research carrier. Martin also gave me great help in administrative procedures for my small family during the time we stayed in Lausanne.

I am grateful to all members of my committee for their reading and evaluating my thesis. Special thanks to David Traum for his helpful comments.

Some years ago, I read a preface from a friend who had just finished his PhD in the HMI group before I arrived. He mentioned that HMI is the best part of living in the Netherlands. I highly appreciate his comment after working for four years in the group. I thank Charlotte Bijron and Alice Vissers-Schotmeijer for their administrative support, Lynn Packwood for proofreading almost all my manuscripts including this thesis, Rieks op den Akker for his advice on the dialogue research topic and helpful comments on this thesis, Dennis Hofs and Boris van Schooten for their collaboration on practical issues relevant to the main topic of this thesis, Hendri Hondorp for his technical support. Many thanks to all other HMI colleagues and friends (especially Natasa Jovanovic, Ronald Poppe, Dolf Trieschnigg, Martijn van Otterlo, and Marijn Huijbregts) for their help and fruitful discussions about work and life in the Netherlands. Special thanks are devoted to my colleagues in the CHIM cluster.

My original schedule in Europe was only for six months for my Master's project. It turns out that I have been staying at least 12 times longer (six years). During that time, I have received great help from, and have had enjoyable discussions with, my

Vietnamese friends in Switzerland (Toàn, Việt, Vũ, Huy, and others), France (Đạt), Vietnam (Bình, Chi, Dương), and the Netherlands (Châu, Cương, Duy, Giang-Chi, Hà-Hạnh, Hà-Hương, Hiển-Như, Hoà, Kim-Vân Anh, Bình Minh, Đức Minh, Nhi, Phong, Phương-Hà, So, Sơn, Thắng-Mai, Tú-An). Especially, I would like to thank Vũ Xuân Hạ for all fruitful discussions during which we have been talking about research and life since we both worked for the same company in Vietnam and continued to work in the same office at EPFL. I also want to thank my close friend Vũ Chí Kiên for his non-research help and advice since the first time I came to the Netherlands and before that when we both studied in the same English course in 1994.

I am indebted for great support from my family in Vietnam. I deeply thank my parents in law for their constant care to my small family. I am very grateful to my mother, my brother, my older sister, and my younger sister for their great love and belief to the progress in my life. Their encouragement makes me stronger and more confident.

Most importantly, I want to thank my wife for her love and extreme patience as well as her daily care for me and our children Hiếu and Thư. This thesis is dedicated to them.

Enschede, September 2008
Trung H. Bui

Contents

List of Acronyms

AR	Affect Recognition	79
AS	Action Selector	104
ADS	Affective Dialogue System	99
ASR	Automatic Speech Recognition	136
BN	Bayesian Network	21
DBN	Dynamic Bayesian Network	99
DDN	Dynamic Decision Network	99
DIS	Dialogue Information State	104
DM	Dialogue Manager	135
DS	Dialogue System	100
GDN	Generic Dialogue Node	47
HMI	Human Media Interaction	8
MDP	Markov Decision Process	82
MDS	Multimodal Dialogue System	80
NLG	Natural Language Generation	80
NLU	Natural Language Understanding	80
PBVI	Point-Based Value Iteration	37
POMDP	Partially Observable Markov Decision Process	163
RDPM	Rapid Dialogue Prototyping Methodology	163
SDS	Spoken Dialogue System	82
TTS	Text To Speech	80
VI	Value Iteration	99
VSM	Vector Space Model	68
WoZ	Wizard of Oz	133

List of Figures

List of Tables

Chapter 1

Introduction

Look Dave, I can see you're really upset about this. I honestly think you ought to sit down calmly, take a stress pill, and think things over.

Excerpt from the Kubrick's science fiction film: "2001: A Space Odyssey"

The HAL 9000 computer character is popular in the speech and language technology research field since his capabilities can be linked to different research topics of the field such as speech recognition, natural language understanding, lip reading, natural language generation, and speech synthesis [88, chap. 1]. This artificial agent is often referred to as a dialogue system, a computer system that is able to talk with humans in a way more or less similar to the way in which humans converse with each other.

Furthermore, HAL was affective[1]. He is able to recognize the affective states of the crew members through their voice and facial expressions, and to adapt his behavior accordingly. HAL can also express emotions, which is explained by Dave Bowman, a crewman in the movie:

> Well, he acts like he has genuine emotions. Of course, he's programmed that way to make it easier for us to talk with him.

Precisely, HAL is an "ideally" Affective Dialogue System (ADS), a dialogue system that has specific abilities relating to, arising from, and deliberately influencing people's emotions [123].

Designing and developing ADSs have recently received much interest from the dialogue research community [7]. A distinctive feature of these systems is affect modeling. Previous work was mainly focused on showing system's emotions to the user in order to achieve the designer's goal such as helping the student to practice nursing tasks [77] or persuading the user to change their dietary behavior [56]. A challenging

[1]We use the term "emotional" and "affective" interchangeably as adjectives describing either physical or cognitive components of the interlocutor's emotion [123, pg. 24].

problem is to infer the interlocutor's affective state (hereafter called "user") and to adapt the system's behavior accordingly.

Solving this problem could enhance the adaptivity of a dialogue system in many application domains. For example, in the information seeking dialogue domain, if a dialogue system is able to detect the critical phase of the conversation which is indicated by the user's vocal expressions of anger or irritation, it could determine whether it is better to keep the dialogue or to pass it over to a human operator [15]. Similarly, many communicative breakdowns in a training system and a telephone-based information system could be avoided if the computer was able to recognize the affective state of the user and to respond to it appropriately [105]. In the intelligent spoken tutoring dialogue domain, the ability to detect and adapt to student emotions is expected to narrow the performance gap between human and computer tutors [17].

This thesis addresses this problem from an engineering perspective using Partially Observable Markov Decision Process (POMDP) techniques and a Rapid Dialogue Prototyping Methodology (RDPM). We argue that POMDPs are suitable for use in designing affective dialogue management models for three main reasons. First, the POMDP model allows for a realistic modeling of the user's affective state, the user's intention, and other (user's) hidden state components by incorporating them into the state space. Second, recent dialogue management research [127, 142, 177, 187] has shown that a POMDP-based Dialogue Manager (DM) is able to cope well with uncertainty that can occur at many levels inside a dialogue system from speech recognition, natural language understanding to dialogue management. Third, the POMDP environment can be used to create a simulated user which is useful for learning and evaluation of competing dialogue strategies [147].

In the following, we first present the research context. We then describe briefly the goals and the main contributions of the thesis. Finally, we give an outline of the remaining chapters.

1.1. Research context: The ICIS project

The work presented in this thesis has been mainly developed in the framework of the Computational Human Interaction Modeling (CHIM) research cluster of the Interactive Collaborative Information Systems (ICIS) project[2]. The main goal of the project is to develop better techniques for making complex information systems more intelligent and supportive in decision making situations.

In the scope of the CHIM research cluster, we aim to develop a multimodal system framework for research in well-defined professional task environments such as the crisis management domain or the air traffic control domain. In these domains, users often experience stress and critical task demands. Our framework, therefore, puts emphasis on the adaptive behavior of the system in recognizing and responding to the user's affective states. Further, robustness of the system toward environment noise also needs to be taken into consideration [66]. Our multimodal system (Fig. 1.1) allows users to interact with it in an action cycle style. Input recognition modules detect

[2]ICIS is sponsored by the Dutch government under the contract BSIK 03024

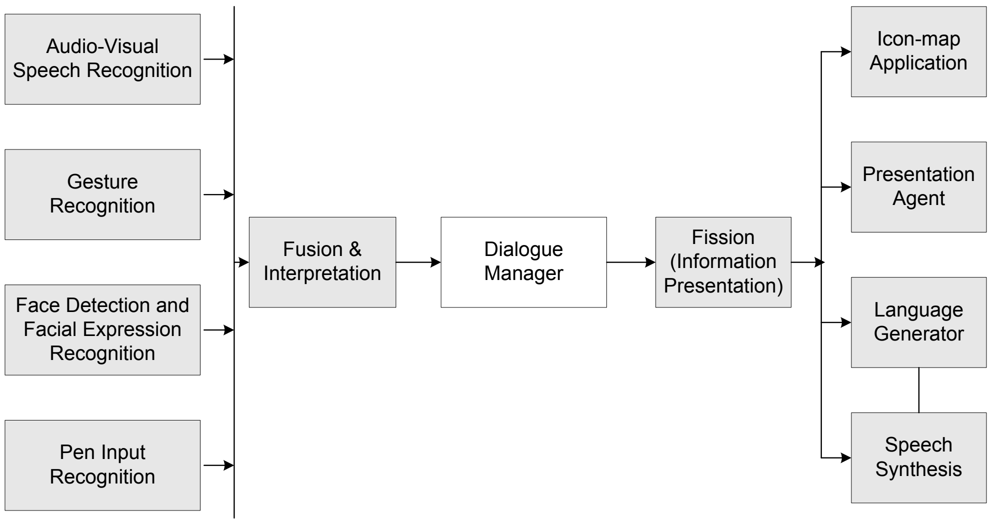


Figure 1.1: Related research topics (boxes in gray color) in the CHIM cluster

the user's speech, lip movement, affective state (through facial expressions or vocal features), pose gestures, and pen input (writing and sketches). These inputs are then preprocessed by the fusion module and are sent to the DM. The DM selects an appropriate system action and sends it to the user through output generation modules where the system action is represented as text, speech, or icon maps or in a mixed form. Modules in gray color (Fig. 1.1) are studied by our colleagues in the CHIM cluster. Section 2.2 will discuss in detail about the architecture of this multimodal system.

1.2. Goals of the Thesis

As mentioned at the beginning of this chapter, the challenging problem in the design of an ADS is to infer the user's affective state and adapt the system's behavior accordingly. This thesis addresses this problem by introducing a dialogue management system which is able to act appropriately by taking into account some aspects of the user's affective state. The computational model used to implement this system is called the *affective dialogue model*. Concretely, our system processes two main inputs, namely the observation of the user's action (e.g., dialogue act) and the observation of the user's affective state. It then selects the most appropriate action based on these inputs and the context. In human-computer dialogue, building this sort of system is difficult because the recognition results of the user's action and affective state are ambiguous and uncertain. Furthermore, the user's affective state cannot be directly observed, and usually changes over time. Therefore, an affective dialogue model should take into account basic dialogue principles, such as turn-taking and grounding, as well as dynamic aspects of the user's affect.

In short, the central goal of this thesis is to develop *a computational model for implementing a robust dialogue manager that is able to adapt its strategies accordingly given observations (with uncertainty) of the user's action and affective state.* This goal is fulfilled by investigating the following research issues:

- Development of a dialogue management framework for traditional (i.e., non-affective) dialogue systems;
- Enhancement of the framework to adapt the dialogue strategy based on the inferred user's affective state.

This stepwise division is appropriate if we consider that an ADS first needs to fulfill requirements in design of a dialogue system in general. The first issue is presented in Chapters 3 and 4. The second one is presented in Chapters 5 and 6.

1.3. Contributions of the Thesis

The following is a summary of major contributions of the thesis. More details about these contributions are reported at the end of Chapters 3, 4, 5, and 6.

1. The most important contribution of the thesis is the tractable hybrid DDN-POMDP method presented in Chapter 6. The distinctive feature of proposed method (compared with other POMDP-based dialogue management methods from the literature) is the ability to handle frame-based dialogue problems with hundreds of slots and hundreds of slot values.
2. The second contribution is the factored POMDP approach for affective dialogue management (Chap. 5). The proposed approach illustrates that the POMDPs are an attractive model for building affective dialogue systems. Further, various technical issues of using the POMDP technique for developing dialogue management are empirically examined, especially scalability problems.
3. The third contribution is the approach to developing multimodal interfaces for multi-application systems which are dialogue systems that allow the user to navigate between a set of applications (Chap. 4). The proposed approach provides a promising framework for designers and developers to implement a dialogue system that is able to handle a large number of applications smoothly and transparently.
4. The fourth contribution is involved in the design and development of the Rapid Dialogue Prototyping Methodology (RDPM) for a quick production of frame-based dialogue models and the associated dialogue-driven interfaces (Chap. 3). My own contributions are mostly involved in extending the RDPM proposed by *Rajman et al.* [134, 135], which was originally used for implementing finite-state spoken dialogue models. In particularly, I invented the solution table and developed the dialogue strategies and the Wizard of Oz (WoZ) Generator. The

usability of the RDPM has been validated through the implementation of several prototype dialogue systems [28, 102, 136].

Additional contributions In the framework of this thesis we have developed several software toolkits that are useful for the practical development of spoken and multimodal dialogue systems: (i) the POMDP toolkit for the development of POMDP-based dialogue managers[3]; (ii) the DDN-POMDP dialogue manager module for the ICIS-CHIM multimodal system demonstrator[4]; and (iii) the RDPM toolkit for a quick production of frame-based dialogue models and their associated dialogue-driven interfaces.

1.4. Outline

The remaining part of the thesis is organized as follows:

Chapter 2 presents the essential background, definitions, and state-of-the-art work that are relevant to the topic of this thesis. Sections 2.1, 2.2, and 2.3 present terminologies related to the research on dialogue systems and four popular state-of-the-art dialogue management approaches. This section is relevant for all the chapters of the thesis. Section 2.4 describes three issues that are important for designing and developing ADSs. This section is relevant for Chapters 5 and 6. Section 2.5 is about the theory of POMDPs and how to find an optimal policy using the simplest exact algorithm. The background is necessary for the work described in Chapters 5 and 6. Section 2.6 discusses the state-of-the-art POMDP algorithms from the literature. It is relevant to Chapters 5 and 6.

Chapter 3 presents the RDPM framework for the development of frame-based dialogue models for single-application systems. We first present core components of the methodology. Second, we illustrate the usability of the methodology through the prototype INSPIRE Smart Home system. Third, we present our preliminary work to extend the methodology for multimodal dialogue models. This chapter is written based on *Bui et al.* [28] and part of the work that is related to my own contribution reported in [102, 136].

Chapter 4 presents a novel approach to developing interfaces for multi-application systems. We first describe in detail the main idea of the approach. We then present a scenario example for producing a dialogue system accessing ten applications in the ICIS domain. This chapter is written based on *Bui et al.* [30].

[3]http://wwwhome.cs.utwente.nl/∼hofs/pomdp/
[4]http://hmi.ewi.utwente.nl/icis/demonstrator/

Chapter 5 presents a factored POMDP approach to affective dialogue management. It is composed of two main parts. The first part is about the description of an affective dialogue model. The second part is the evaluation of the model and the comparison with other techniques. Part of this chapter is written based on *Bui et al.* [31, 33].

Chapter 6 presents a tractable hybrid DDN-POMDP approach to affective dialogue management. The first part of this chapter is about the description of the approach. The second part is about the experiments and evaluation of the method through a multi-slot route navigation example. Part of this chapter is written based on *Bui et al.* [32, 34].

Chapter 7 summarizes the main points of the thesis and discusses future directions.

Chapter 2

Background and definitions

2.1. Introduction

Dialogue is conversation between two or more agents, be they human or computer. Research on dialogue usually follows two main directions: *human-human dialogue* and *human-computer dialogue.* The former is relevant to the study of *discourse analysis* and *conversation analysis* [see 99, chap. 6]. This thesis focuses on human-computer dialogue which is involved in a *Dialogue System (DS)*, a computer system that is able to talk with a human (hereafter called “user"). In the following, we describe briefly key concepts and related work from studies of human-human dialogue that are particularly important for designing DSs:

- **Speech act and dialogue act**. The term *speech act* originates from Austin's work [11]. His point of view is that an utterance in a dialogue is an *action* performed by the speaker. Speech acts, therefore, are considered as performative verbs such as *name*, *second*, and *best.* Speech acts are mainly referred to as *illocutionary acts* which Austin defined as acts of asking, answering, making a request, making a promise. Searle further developed the theory of speech acts described in his influential book [150]. In designing DSs, the notion of speech act is expanded, and Bunt coined the term *dialogue act.* Dialogue acts refer to *functional units used by the speaker to change the dialogue context* [35]. Concretely, a dialogue act is composed of three aspects: the *utterance form* (e.g. “Is it raining?"), the *communicative function* (e.g. “Yes-No-Question"), and the *semantic content* (e.g. the proposition “it is raining"). Concepts similar to dialogue acts were proposed by a number of researchers who aimed to annotate dialogue corpora and to generalize the dialogue management framework: *conversation acts* [167], *conversational moves* [37], and *dialogue moves* [51].

- **Turn-taking**. In a conversation, two participants exchange turns in sequence (one talks, stops; another starts, talks, stops and so on) like two persons playing tennis. Each turn is composed of one or more utterances performed by the speaker to addressees. Turn transfers are assumed to occur at certain points called Transition Relevance Places (TRPs) and not at others [146]. Levinson[99, pp. 296-297] described two important empirical facts about turn-taking in human-human dialogue: (i) less than five percent of the speech stream is delivered in overlap and (ii) the gap between two consecutive turns is on average only a few tenths of one second. *Sacks et al.* [146] suggest that the turn-taking mechanism can be described by a set of rules, called the SSJ model, which are simplified as follows: (i) if C (current speaker) selects N (next speaker) then C must stop speaking and N must speak next; (ii) if C does not select N, then any other party may self-select and the first one will take the next turn; and (iii) if C has not selected N and no other party self-selects then C may continue. Although the SSJ model has been widely used as a normative description of interactive systems, there are cases where the simultaneous expressive behavior of speaker and listener are important. For example, *Nijholt et al.* [115] have recently discussed the important role of this behavior in three non-verbal interactive applications (an interactive dancer, an interactive conductor, and an interactive trainer) developed at the Human Media Interaction (HMI) group. In these applications, expressive behavior of a virtual human has to be synchronized with that of the user.

- **Grounding**. In conversation, both the speaker and the addressee need to establish a common ground [162] in order to ensure that the addressee understands clearly the speaker's meaning and intention. *Clark and Schaefer* [47] proposed a significant formal model, called the *contribution model*, for detecting and repairing communication errors in human-human conversations. *Traum* [166] developed an online reformulation of the contribution model, called the *grounding acts model*, which was used in developing a collaborative conversational agent in the TRAINS project [5]. *Cahn and Brennan* [36] formalized and extended the contribution model to explicitly represent the system's private model. *Paek and Horvitz* [119] proposed a formalization of grounding based on inference and decision making under uncertainty. This model was used to develop two prototype dialogue systems: *Presenter* and *Bayesian Receptionist* [118].

There are many types of DSs that are classified by modality, device, initiative, application, and task complexity [53]. In the framework of this thesis, I am particularly interested in goal-oriented (or task-oriented) DSs. The main objective of a goal-oriented DS is to cooperate (and partly, collaborate) with the user to help the user achieve their goal. Among goal-oriented DSs, we distinguish two popular types: (i) *Spoken Dialogue Systems (SDSs)* and (ii) *Multimodal Dialogue Systems (MDSs).*

A SDS, also called *conversational agent*, is a dialogue system that understands and responds to the user using speech. A good survey of SDSs is described in detail in *McTear* [107, 108].

An MDS is a dialogue system that processes two or more combined user input modes - such as speech, pen, touch, manual gestures, gaze, and head and body movements - in a coordinated manner with multimedia system output [117]. We view an SDS as a particular type of MDS.

In the following sections, we first give an overview of an MDS. We then present the related dialogue management approaches from literature.

2.2. Overview of a dialogue system

An MDS usually consists of the following components: *Input*, *Fusion*, *Dialogue Manager (DM)* and *Knowledge Sources*, *Fission*, and *Output*. Figure 2.1 shows a conceptual architecture of an MDS designed and partially implemented in the framework of the ICIS project.

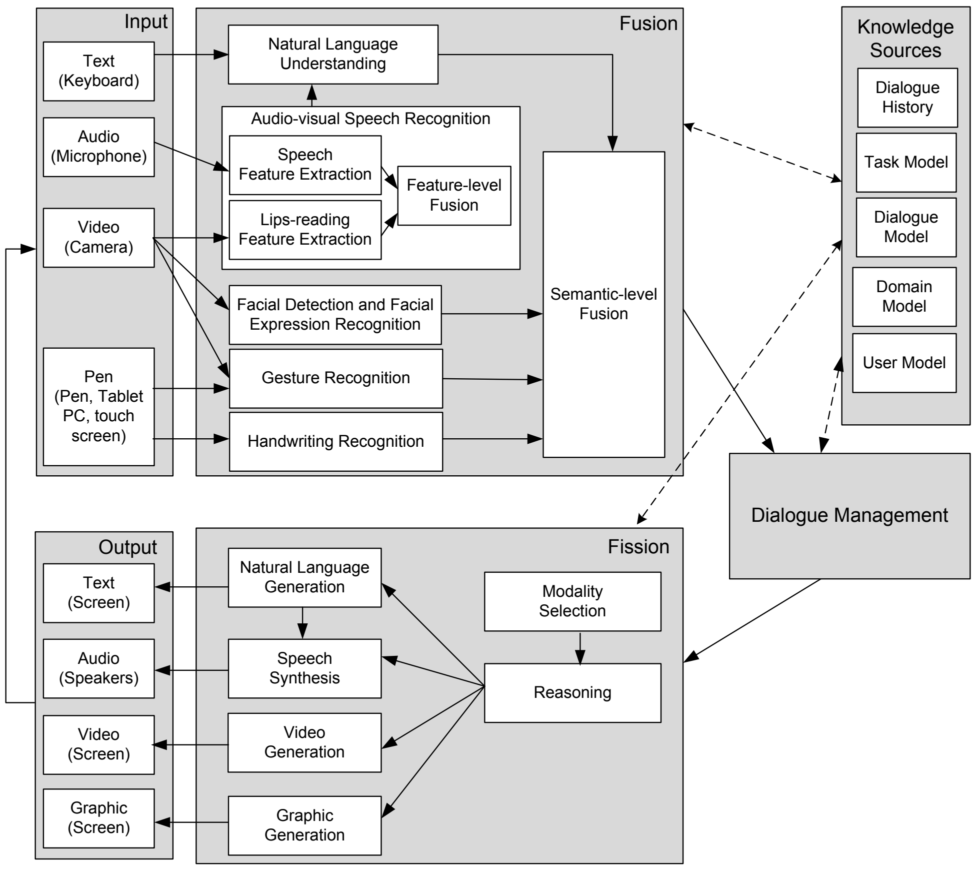


Figure 2.1: Conceptual architecture of a multimodal dialogue system designed and partially implemented in the ICIS project. In our current prototype, the system was implemented as a distributed system where all modules exchange messages through an interaction middleware.

2.2.1. Input

Inputs of an MDS are a subset of the various modalities such as speech, pen, facial expressions, gestures, and gazes. Two types of input modes are distinguished: *active input modes* and *passive input modes* [117]. *Active input modes* are the modes that are deployed by the user intentionally as an explicit command to the computer such as speech. *Passive input modes* refer to naturally occurring user behavior or actions that are recognized by a computer (e.g., facial expressions). They involve user input that is unobtrusively and passively monitored, without requiring any explicit command to a computer [117]. Examples of MDSs that combine multiple input modalities are:

- Speech and (hand, pen, or pointing) gesture [42, 48, 87, 102, 154],
- Speech and haptics [77],
- Speech, gestures, and facial expressions [173].

2.2.2. Fusion

Information from various input modalities is extracted, recognized and fused. Fusion processes the information and assigns a semantic representation which is eventually sent to the dialogue manager. In the context of MDSs, two main levels of fusion are often used: *feature-level fusion*, *semantic-level fusion*. The first one is a method for fusing low-level feature information from parallel input signals within a multimodal architecture (for example, in Fig. 2.1, feature-level fusion happens between speech and lip-reading input). The second one is a method for integrating semantic information derived from parallel input modes in a multimodal architecture (for example, in Fig. 2.1, semantic-level fusion happens between input modality action recognition modules such as speech and gesture).

Another related work on low-level fusion is *sensor fusion*. Sensor fusion combines sensory data from disparate sources to gain a more accurate information[1].

Semantic-level fusion is usually involved in the DM and needs to consult the shared knowledge sources (see Fig. 2.1). Three typical semantic fusion techniques are used from the literature: *frame-based fusion*, *unification-based fusion*, and *hybrid symbolic/statistical fusion*. Frame-based fusion is a method for integrating semantic information derived from parallel input modes in a multimodal architecture, which has been used for combining speech and gesture (e.g. *Vo and Wood* [171]). Unification-based fusion is a logic-based method for integrating partial meaning fragments derived from input modes into a common meaning representation during multimodal language processing. Compared with frame-based fusion, unification-based fusion derives from logic programming, and has been more precisely analyzed and widely adopted within computational linguistics (e.g. *Johnston* [86]). Hybrid symbolic/statistical fusion is an approach to combine statistical processing techniques with a symbolic unification-based approach (e.g. Members-Teams-Committee (MTC) hierarchical recognition fusion [180]).

[1]http://en.wikipedia.org/wiki/Sensor_fusion[accessed 2008-04-25]

2.2.3. Dialogue manager

The DM is the core component of a dialogue system. It processes semantic inputs from fusion (either semantic concepts, dialogue acts or communicative intentions) and decides what the system should do next in response to the user in order to fulfil the user's goal.

In the following we briefly describe a number of knowledge sources that are usually used by the Dialogue Manager, Fusion, and Fission. The knowledge sources deployed in DSs are discussed further in *Flycht-Eriksson* [67].

- *Dialogue history.* Dialogue history is composed of a set of utterances (or concepts, dialogue acts) which were uttered by the system and the user from the beginning of a dialogue session to the current turn.
- *Task model.* A precise definition of the task model depends on each application domain. Generally speaking, we can say that the task model is composed of the information the system needs to gather to complete the system's task that fulfils the user's goal. For example, in the information seeking domain, the task model is composed of a set of information pieces that the system needs to collect from the user to execute its task (such as making a database query).
- *Domain model.* A model with specific information about the application domain. For example, in the flight information domain, we can apply constraints to disambiguate input, such as, the departure location and the destination location must be different.
- *User model.* This model may contain relatively stable information about the user that may be relevant to the dialogue such as the user's age, gender, and preferences (user preferences) as well as information that changes over the course of the dialogue, such as the user's goals, beliefs, and intentions (user's mental states).

2.2.4. Fission

Fission is the process of realizing an abstract message through output on some combination of the available channels. The tasks of a fission module is composed of three categories [69]:

- *Content selection and structuring*: the presented content must be selected and arranged into an overall structure.
- *Modality selection*: the optimal modalities are determined based on the current situation of the environment, for example, when the user device has a limited display and memory, the output can be presented in graphic form such as a sequence of icons.
- *Output coordination*: the output on each of the channels should be coordinated so that the resulting output forms a coherent presentation.

2.2.5. Output

Various output modalities can be used to present the information content from the fission module such as speech, text, 2D/3D graphics, avatar, and haptics. Popular combinations of the output modalities are: (1) graphics and avatar, (2) speech and graphics, (3) text and graphics, (4) speech and avatar, (5) speech, text, and graphics, (6) text, speech, graphics, and animation, (7) graphics and haptic, (8) speech and gesture.

2.3. Approaches to dialogue management

Four popular state-of-the-art approaches to design and development of dialogue management models[2] are: *finite state*, *frame-based*, *information state*, and *MDP-based.* These approaches will be described in the next sections. Other approaches are reported in *Bui* [26]. To illustrate the main idea of each approach, we use the restaurant information system, RestInfo, which was developed in the framework of the InfoVox project [136]. The RestInfo DS aims to provide information about the restaurants of the city of Martigny, Switzerland.

2.3.1. Finite state dialogue models

The finite state approach [75, 106, 134] models dialogue as a finite state machine [88, chap. 2] where the nodes correspond to system prompts and the arcs correspond to the user's choices. Transitions between nodes determine all possible paths of the dialogue flow. The interaction is usually system-initiative and all the system's prompts are predetermined. Figure 2.2 shows a simplified model for the Restinfo system. The system interacts with the user to collect pieces of information step by step and to confirm all the collected values before conducting a query to the database. A dialogue session using this model is shown in Table 2.1 where S and U denote the system and the user, respectively.

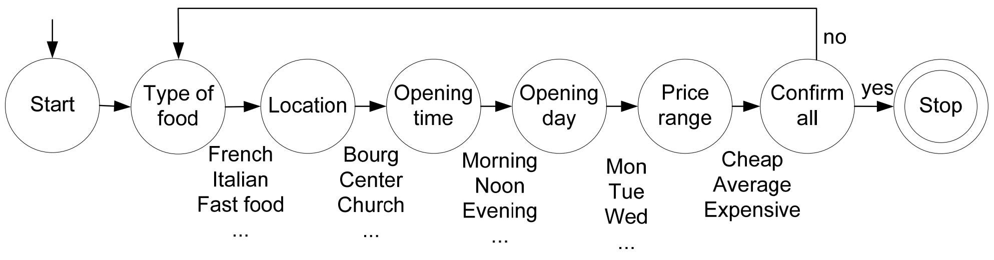


Figure 2.2: Simplified finite-state dialogue model for the RestInfo system

[2]We defer discussion on the POMDP-based dialogue management approach from the literature to Chapters 5 and 6.

	Utterance	Active node
S_1 :	Welcome to the RestInfo service.	
	I can help you to find a restaurant.	Start
	What type of food are you looking for?	Type of food
U_1 :	Italian.	
S_2 :	and what location?	Location
U_2 :	in the center.	
S_3 :	and what time of the day?	Opening time
U_3 :	Evening.	
S_4 :	and which day of the week?	Opening day
U_4 :	Sunday.	
S_5 :	and what price level?	Price range
U_5 :	Average.	
S_6 :	So you want to find an average Italian restaurant in the center on Sunday, in the evening. Is that correct?	Confirm all
U_6 :	Yes.	
S_7 :	Please wait a moment. *[queryDB(?x.info:x.typeOfFood=Italian, x.location=center, x.openingTime=evening, x.openingDay=sunday,x.priceRange=cheap)]* I found “Le café de Martigny". Thank you for using our service.	Stop

Table 2.1: Restinfo dialogue session to illustrate the finite-state approach

Finite state dialogue models are easy to construct and robust to cope with the speech recognition problem. This approach is suitable for simple applications with well-structured tasks. However, this approach is inflexible, unnatural, and difficult to apply to more complex tasks [107].

2.3.2. Frame-based dialogue models

The frame-based approach [19, 21, 45, 73, 83, 143] is able to solve some of the limitations of the finite-state approach. A frame-based dialogue model is represented by a set of frames: each frame is composed of a number of slots (Fig. 2.3). The task of the dialogue manager is considered as a slot-filling or form-filling task. A predefined set of information is gathered in a more flexible way. An example of the dialogue between the system and the user is shown in Table 2.2. Frame-based DSs can handle more flexible user input. For example, the user can provide extra information other than that the system has asked for in the previous turn. Most of the current commercial DSs are frame-based DSs [125].

S_1 :	Welcome to the RestInfo service. What can I do for you? *[focus GDN: start, active GDNs: all]*
U_1 :	Find a restaurant with Italian food near the center. *[typeOfFood=Italian, location=center]*
S_2 :	For Italian as type of food and Center as location. What time? *[implicit_ confirm=true, focus GDN: time, active GDNs: all]*
U_2 :	Sunday evening. *[openingTime=evening, x.openingDay=Sunday]*
S_3 :	For Evening as time and Sunday as day. What price range? *[implicit_ confirm=true, focus GDN: priceRange, active GDNs: all]*
U_3 :	Cheap. *[priceRange=cheap]*
S_4 :	Please wait a moment. *[queryDB(?x.info:x.typeOfFood=Italian, x.location=center, x.openingTime=evening, x.openingDay=sunday, x.priceRange=cheap)]*
S_5 :	I found “Le café de Martigny". What else can I do for you?
U_4 :	No, thanks.
S_6 :	Thank you, good bye.

Table 2.2: RestInfo dialogue session to illustrate the frame-based approach

The original idea of representing knowledge as *frames* was proposed by *Minsky* [111]. One of the oldest frame-based DSs is GUS [19]. GUS was designed to act as a travel agent that helps clients to make a simple return trip to a single city in California. It mainly focuses on language-understanding problems such as that in understanding indirect answers to questions, resolving anaphora. Although the motivational dialogue example is speech-based, the implemented version of the system is text-based and does not have a separate dialogue management module. However,

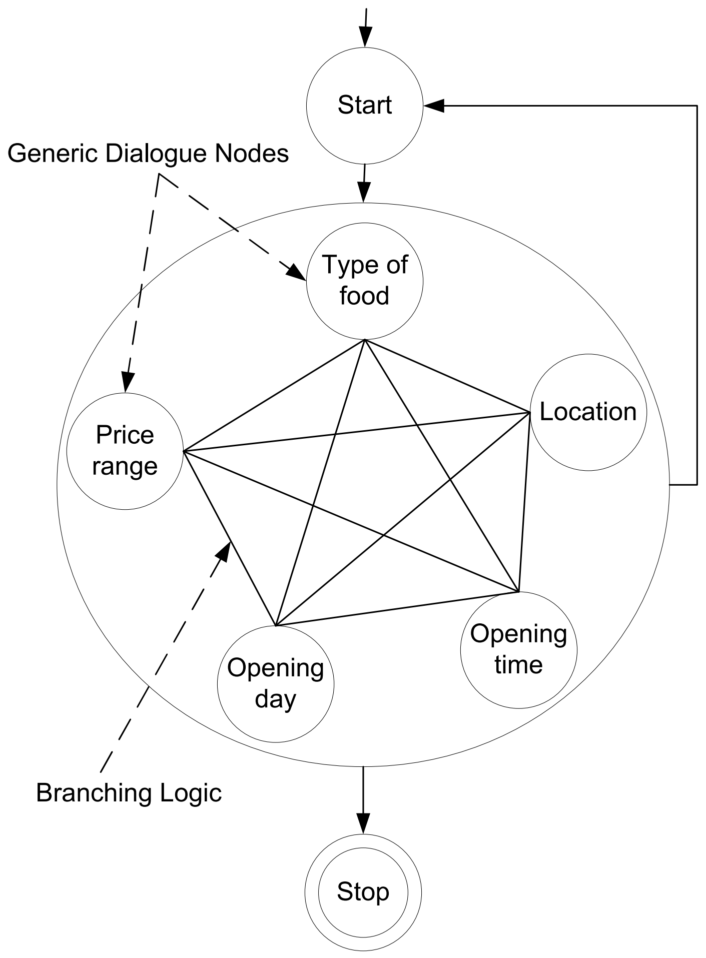


Figure 2.3: Frame-based dialogue model for the RestInfo system based on Bui et al. [27]. Each slot is modeled as a generic dialogue node. See Chapter 3 for a further explanation.

the idea of using frames for the reasoning component is explained clearly and the idea of using an agenda for the system control is mentioned. Significant efforts to develop spoken and multimodal frame-based dialogue managers are contributed by a number of authors [21, 45, 73, 83, 143]. Chapters 3 and 4 present our contribution to rapid prototyping for designing spoken and multimodal frame-based DSs.

2.3.3. Information state dialogue models

The information state theory of dialogue consists of five main types of unit: a description of *informational components*, *formal representations* of the informational components, a set of *dialogue moves*, a set of *update rules*, and an *update strategy* [95, 165]. These units [165] are briefly described as follows:

- The informational components include aspects of common context and internal motivational factors such as participants, common ground, linguistic and intentional structure, obligations and commitments, beliefs, intentions and user models.
- The formal representations of the information components might be implemented as lists, sets, typed feature structures, records, discourse representation structures, propositions or modal operators within a logic, et cetera.
- Dialogue moves will be used to trigger the update of the information state. These moves are correlated with externally performed actions such as natural language utterances being realized by the Natural Language Generation (NLG) module.
- The update rules govern the updating of the information state given various conditions of the current information state and performed dialogue moves, including a set of selection rules, that license choosing a particular dialogue move to perform given conditions of the current information state.
- The update strategy decides which rule(s) to apply at a given point from the set of applicable ones. This strategy can range from something as simple as “pick the first rule that applies" to more sophisticated arbitration mechanisms, based on game theory, utility theory, or statistical methods.

Table 2.3 shows an example of the interaction between the user and the Restinfo system formulated based on the information state theory. The information state after turn U_1 is shown in Table 2.4. Dialogue moves include: *greet, offer, ask, inform, answer*, and *quit*.

The information state approach is intended to be a unified model for designing dialogue management. A wide range of applications can be modeled using this approach from simple applications such as Restinfo to more complex applications (e.g., [64, 168]).

S_1 :	Welcome to the Restinfo service. What can I do for you?
	[greet, offer]
U_1 :	I am looking for a restaurant.
	[ask(x.info)]
S_2 :	What type of food are you looking for?
	[ask(x.typeOfFood)]
U_2 :	Italian food near the center.
	[answer(x.typeOfFood=Italian, x.location=center)]
S_3 :	and what time do you prefer?
	[ask(x.openingTime)]
U_3 :	Sunday evening.
	[answer(x.openingTime=evening, x.openingDay=sunday)]
S_4 :	and what about the price range: cheap, average, or expensive?
	[ask(x.priceRange)]
U_4 :	Cheap.
	[answer(x.priceRange=cheap)]
S_5 :	Please wait a moment.
	[queryDB(?x.info:x.typeOfFood=Italian, x.location=center, x.openingTime=evening, x.openingDay=sunday, x.priceRange=cheap)]
S_6 :	I found "Le café de Martigny". What else can I do for you?
	[answer(x.info), offer]
U_5 :	No.
	[answer(no)]
S_7 :	Thank you, good bye.
	[quit]

Table 2.3: Restinfo dialogue session to illustrate the information-state approach

$$\left|\begin{array}{lcl}
\text{PRIVATE} & = & \left|\begin{array}{lcl}
\text{AGENDA} & = & \langle\rangle \\
\text{PLAN} & = & \left|\begin{array}{l}
\text{ask(?x.typeOfFood)} \\
\text{ask(?x.location)} \\
\text{ask(?x.openingTime)} \\
\text{ask(?x.openingDay)} \\
\text{ask(?x.priceRange)} \\
\text{queryDB(?x.info)}
\end{array}\right| \\
\text{BEL} & = & \{\}
\end{array}\right| \\
\text{SHARED} & = & \left|\begin{array}{lcl}
\text{COM} & = & \\
\text{QUD} & = & \langle\text{?x.info}\rangle \\
\text{LU} & = & \left|\begin{array}{lcl}
\text{SPEAKER} & = & \text{user} \\
\text{MOVES} & = & \langle\text{ask(?x.info)}\rangle
\end{array}\right|
\end{array}\right|
\end{array}\right|$$

Table 2.4: Restinfo information state after U_1 (adapted from [94])

2.3.4. MDP-based dialogue models

A key limitation of the information state dialogue models (as well as finite-state and frame-based dialogue models) is that designers have to define update rules, which can be very time-consuming and labor-expensive. This triggered the quest to find mechanisms to learn a good dialogue strategy from dialogue corpora automatically. The Markov Decision Process (MDP) based dialogue models were proposed. The idea is to formulate dialogue as an MDP and then use reinforcement learning techniques to find an optimal policy (i.e. dialogue strategy). Figure 2.4 shows an MDP-based dialogue model for the RestInfo system adapted from the Pietquin model [126]. The state space is composed of variables for slots and variables to monitor the current state of the dialogue such as grounding (*Status*), Automatic Speech Recognition (ASR) confidence level (*ASR confidence*), and the number of retrieved records from the database (*number of DB records*). The action set is composed of slot independent actions (such as *greet*, *ask all*, *confirm all*, *query database*, and *quit*) and slot dependent actions such as (*ask slot X*, *confirm slot X*, and *relax slot X*). The reward function is defined based on the number of system and user turns and the task completion [126, pg. 200].

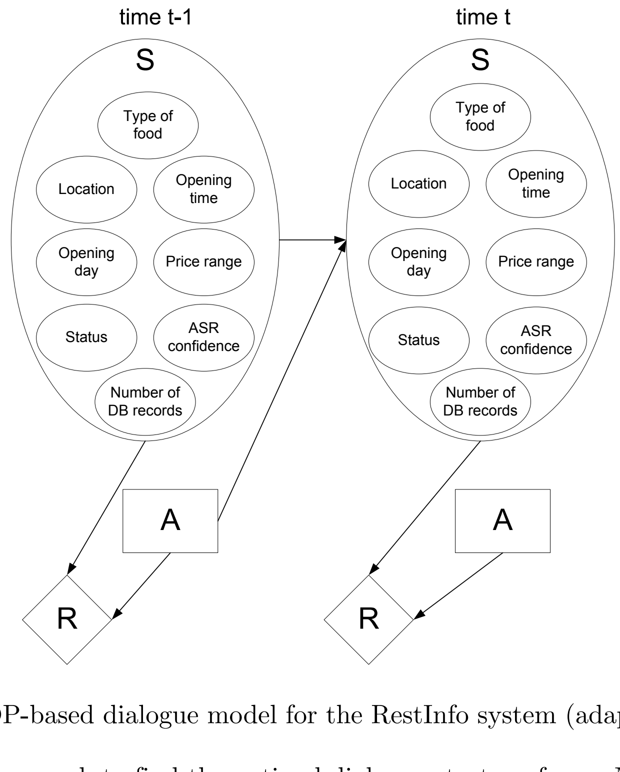


Figure 2.4: MDP-based dialogue model for the RestInfo system (adapted from [126])

A popular approach to find the optimal dialogue strategy for an MDP-based dialogue model is to use user simulation techniques [98, 126]. The task is composed of a two-phase approach (Fig. 2.5): (i) a simulated user is first trained (using supervised learning techniques) on a small human-computer dialogue corpus to learn responses of a real user given the dialogue context; (ii) the learning DM then interacts with this simulated user in a trial-and-error manner to learn an optimal dialogue strategy.

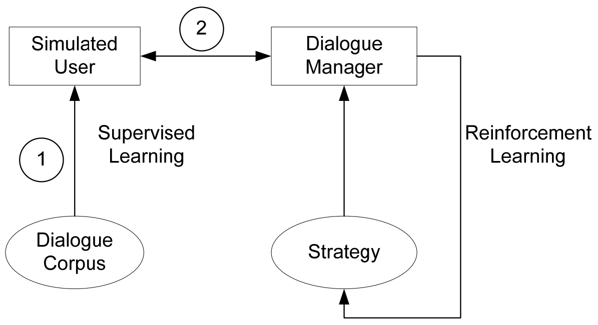


Figure 2.5: Two-phase approach for dialogue strategy learning

2.4. Toward affective dialogue systems

Emotion has been taken into consideration in designing DSs since the start of the 1970s. *Artificial Paranoia*, developed by *Colby et al.* [49], was the first text-based DS that could express `fear` and `anger` based on keywords extracted from the user's input. But only recently, the design and development of Affective Dialogue Systems (ADSs) have received much interest from the dialogue research community [7].

A DS that can detect the user's affective state could be beneficial in many application domains. For example, in the information seeking dialogue domain, if a DS is able to detect the critical phase of the dialogue which is indicated by the user's vocal expressions of anger or irritation, it could determine whether it is better to keep the dialogue or pass over to a human operator [15]. Similarly, *Martinovsky and Traum* [105] showed that many communicative breakdowns in a training system and a telephone-based information system could be avoided if the computer was able to recognize the emotional state of the user and to respond to it appropriately. In the intelligent spoken tutoring dialogue domain, the ability to detect and adapt to student emotions is expected to narrow the performance gap between human and computer tutors [17].

A DS that can express emotions appropriately based on social norms and the dialogue context is also advantageous [77, 122]. The main motivations behind this work are based on the empirical studies of *Reeves and Nass* [138]. They claimed that users tend to apply social norms to computers. Showing system's affect to the users when appropriate, therefore, could enhance users' satisfaction and preferences. For example, the *Cosmo* pedagogical agent intentionally expresses emotions to encourage the student in problem-solving tasks. *Greta* [122] is provided with a personality and a social role that allow her to decide whether to show her emotion or not depending on the current dialogue context. *INES* [77] exploits different tutoring strategies and expresses empathic emotions toward the student depending on whether the student is confident or insecure.

Beyond development issues of an MDS, three important issues need to be taken into account for the design of an ADS:

- How does the system recognize the user's affect?
- How does the system incorporate the user affect model?
- How does the system express emotion during a conversational session with the user?

These issues are presented in Sections 2.4.1, 2.4.2, and 2.4.3, respectively. We also describe the related work that is particularly useful in the DS development framework.

2.4.1. Affect recognition

The user's affect can be recognized or inferred from speech [9, 82, 181], facial expressions [151], physiological signals [124, 137], or multiple modalities [57]. A good survey of different approaches to recognize affect is presented in *Zeng et al.* [186]. Which modality is best to use for the recognition task depends on the application domain. For example, in a telephone-based DS, the obvious source is from the user's vocal input. In an in-car DS, besides speech we can take advantage of facial expressions (detected from the cameras installed inside the car) and physiological signs recognized through devices which can be quite easily set up inside the car system [65].

Figure 2.6 shows an emotion recognition system proposed by *Lee and Narayanan* [96] which is able to recognize two emotional states: *negative* and *non-negative.* The user's speech input is first processed by the feature extraction module. The acoustic features (fundamental frequency (F_0), energy, duration, first and second formant frequencies and their bandwidths) are then combined with the lexical information (emotionally salient words) and discourse information (five speech act labels of the user's response to the system: *rejection*, *repeat*, *rephrase*, *ask-start over*, and *none of the above*) by the emotion recognizer module. The best classification result for both male and female speakers is about 90%.

2.4.2. Affective user modeling

When building an ADS, an interesting question is which emotion category should we use to model the user's affect and which user's affective states should we select? It depends on the application domain. For example, in the tutoring domain, *D'Mello et al.* [57] showed that the user (in this case the student or learner) rarely experienced sadness, fear, or disgust. Therefore, modeling the user's affect using six basic emotions (fear, anger, happiness, sadness, disgust, and surprise), on which most of the state-of-the-art emotion classification work is focusing, is not a good choice for this domain.

The task of integrating the user affective state model into the system is called *affective user modeling.* Most of the affective user models categorize the user's affective

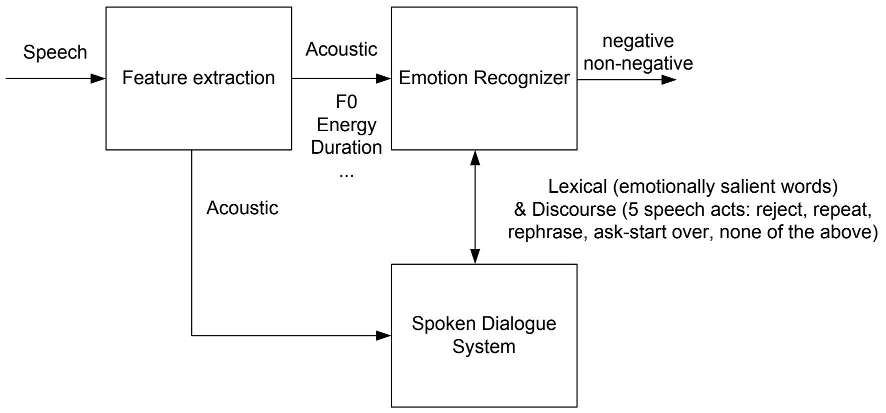

Speech
Feature extraction
Acoustic
F0
Energy
Duration
...
Emotion Recognizer
negative
non-negative
Acoustic
Lexical (emotionally salient words)
& Discourse (5 speech acts: reject, repeat, rephrase, ask-start over, none of the above)
Spoken Dialogue System

Figure 2.6: Emotion recognition module proposed by *Lee and Narayanan* [96]

state based on a subset of 22 emotion types in the OCC model developed by *Ortony et al.* [116] such as *Conati* [50], *Elliott et al.* [60], *Katsionis and Virvou* [91], *Martinho et al.* [104] or on dimensional-based models [144] such as *Ball and Breese* [14], *Kort et al.* [93]. These models are then represented using Bayesian Networks (BNs) [121].

Key advantages of using BNs to model the user's affect are: (i) They deal explicitly with uncertainty between the user's affect and their observed behaviors; (ii) The links between nodes are meaningful because we can interpret these links as the causal relations between variables; (iii) The network can be easily extended by adding new behavior nodes and linking them to the most relevant hidden nodes; and (iv) They can handle mixed emotions in a flexible manner [38].

One of the first affective user models was proposed by *Ball and Breese* [14]. The user's emotion and personality are modeled by four variables: *valence*, *arousal*, *dominance*, and *friendliness*. The values of these variables are inferred from the observable user's behaviors such as *speech*, *gesture*, *posture*, and *facial expressions* (Fig. 2.7). This model can also be used for the system to express emotions.

However, Ball's model and other static BN-based models [114] do not represent the temporal evolution of the user's emotional state as well as the causal relationships between the emotional states and the personality states (i.e. *dominance* and *friendliness*). To correct these shortages, Dynamic Bayesian Networks (DBNs) have been proposed to model the user's affect, their causes and the relevant observable behaviors [13, 38, 50, 101] (Fig. 2.8).

For example, *Conati* [50] proposed a model for her application game Prime Climb, where the causal relationships between the user's affective states and their causes are based on the OCC model. The `causes` are composed of: *user's traits* (conscientiousness, agreeableness, extraversion), *user's goals* (avoid falling, succeed by myself, have fun), *user's action outcomes*, and *goals satisfied*. The `affective states` are composed of seven variables: *reproach*, *shame*, *joy*, *negative valence*, *positive valence*, *arousal*, and *engagement*. The `observable behaviors` are composed of two levels:

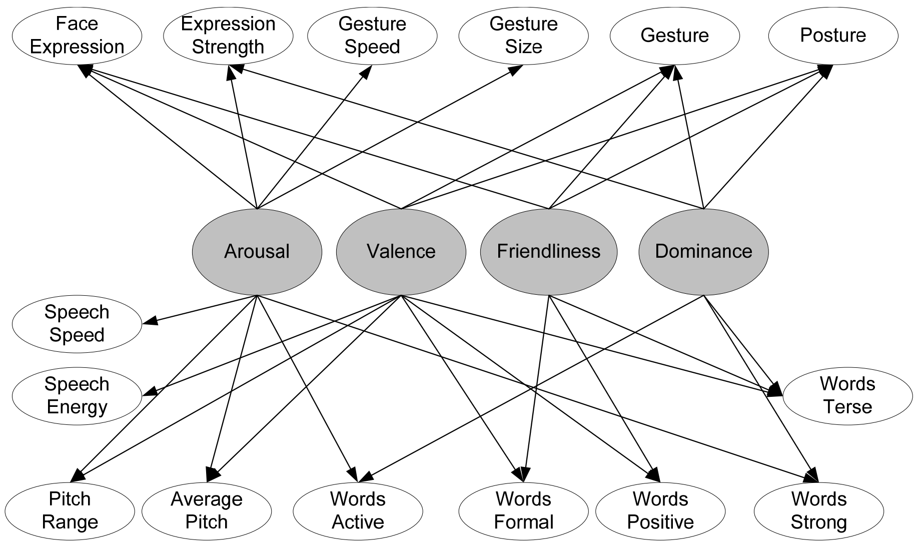


Figure 2.7: Affective user model proposed by *Ball and Breese* [14]

bodily expressions (eyebrow position, skin conductance, and heart rate) and *sensors* (visual based recognizer, Electromyogram (EMG), Galvanic Skin Response (GSR), and heart monitor).

Similarly, in Liao's model [101], which is also used for a real-time stress recognition task, the `causes` are composed of: *context* (complex or simple), *profile* (health, age, skill), *goal* (important, not important), *workload* (high, normal, low). The `affective states` are composed of *stress*, *fatigue*, and *nervous*. The `observable behaviors` are composed of: *physical* (eyelid movement, pupil, facial expression, head movement), *physiological* (Electrocardiogram (ECG), Electroencephalogram (EEG), GSR, and General Somatic Activity (GSA), *behavioral* (mouse movement, mouse pressure, and typing speed), and *performance* (response and accuracy).

2.4.3. Affective system modeling and expression

Early DSs such as Artificial Paranoia used a set of simple rules to express emotions toward the user. Much of research interest has focused on developing computational models of emotion for designing and developing agents (believable agents, virtual humans) that can express realistic and complex emotions as humans do [16, 25, 59, 61, 74, 139, 169]. For example, an emotion model, called *ParleE*, developed at the Human Media Interaction Group, University of Twente is shown in Figure 2.9 [25].

ParleE is an MDP-based model developed based on the OCC model [116] and the personality model [140]. Emotional states are generated based on events from a fully observable environment. ParleE is appropriate for modeling multi-agent systems in a virtual world. It was partially integrated into the INES system [77] which allows the

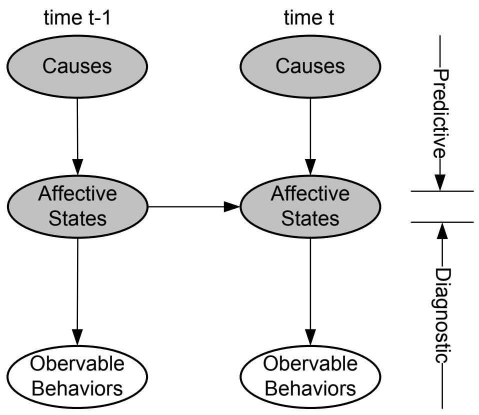


Figure 2.8: High level abstraction of DBN-based affective user models, each node is composed of a set of variables

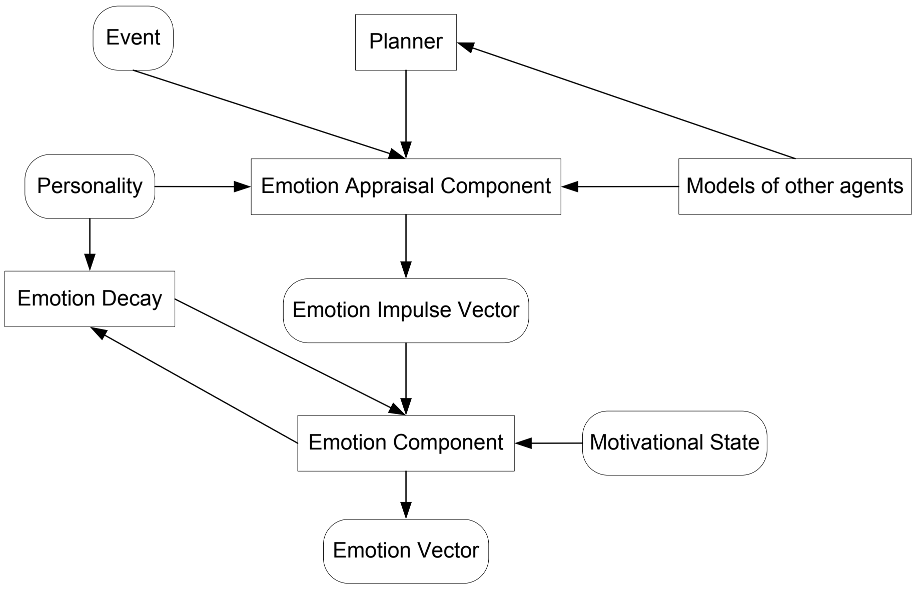


Figure 2.9: Emotion model proposed by *Bui et al.* [25]

system to express four emotion states (*joy*, *distress*, *happy-for*, and *sorry-for*) through a 3D talking head.

One great difficulty of applying ParleE and other emotion models mentioned above directly to the design of ADSs is that these models do not take into account the uncertainty of the user's states such as the uncertainty about the user's emotion, goal, and belief. For example, consider a situation in which the user's affective state is *joy* and the system recognizes it as *distress* and expresses *sorry-for* emotion toward the user by applying the OCC rules. This expression is not appropriate regardless of how realistic the emotions that the system can show are.

2.5. Theory of POMDPs

A Partially Observable Markov Decision Process (POMDP) is a generalization of a Markov Decision Process (MDP) which permits uncertainty regarding the state of the environment. *Howard* [81] described a transition in an MDP as a frog in a pond jumping from lily pad to lily pad. In a POMDP environment, the lily pond is covered by the mist, therefore the frog is no longer certain about the pad it is currently on [112]. Before jumping, the frog can observe information about its current location. This intuitive view is very appropriate to model the affective dialogue management problem as being illustrated in Section 2.5.2.

In a dialogue management context, the agent is the dialogue manager, loosely called, the system (Fig. 2.10). Part of the POMDP environment represents the user's state and user's action. Depending on the design for a particular dialogue application, the rest of the POMDP environment might be used to represent other modules such as the speech recognition and the emotion recognition (see Chap. 5). Because the user's state cannot be directly observed, the agent uses a state estimator (SE) to compute its internal belief (called *belief state*) about the user's current state and an Action Selector (AS) where the policy π is implemented to select actions. The SE takes as its input the previous belief state, the most recent system action and the most recent observation, and returns an updated belief state. The AS takes as its input the agent's current belief state and returns an action that will be sent to the user.

In the following sections, we first describe a basic framework of POMDPs. Second, we present a simple empathic dialogue agent example for the tutoring domain. Third, we describe two main tasks of the agent: *belief updating* and *finding an optimal policy*.

2.5.1. Basic framework

A POMDP is defined as a tuple $\langle S, A, Z, T, O, R\rangle$, where S is a set of states of the environment (usually called *state space*), A is a set of the agent's actions, Z is a set of observations the agent can experience of its environment, T is a *transition function*, O is an *observation function*, and R is a *reward function*. We assume that S, A, Z are finite and that the interaction between the agent and environment follows a sequence of discrete time steps.

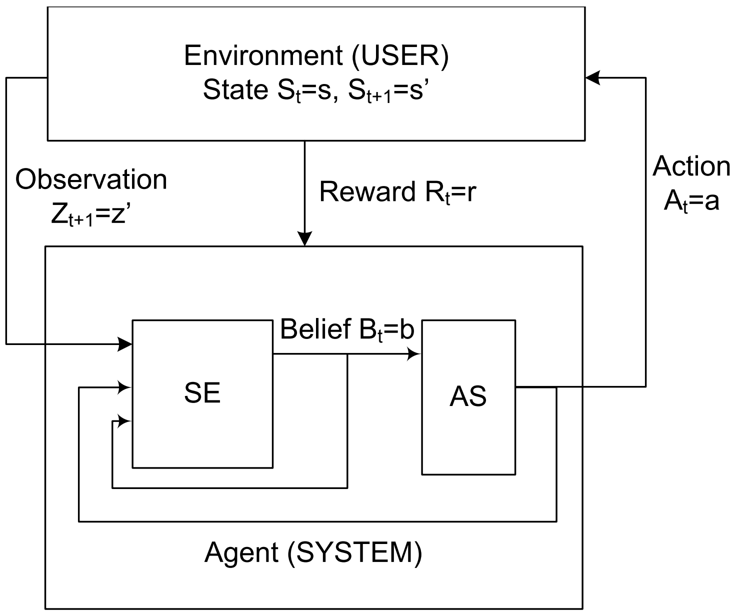


Figure 2.10: Modularized view of the interaction between the dialogue manager and the user in a dialogue management context

Let S_t, A_t, Z_{t+1}, and R_t be random variables taking their values from the sets S, A, Z, and $\mathbb{R}$ (the set of real numbers), respectively. At each time step t, the environment's state is S_t. The agent selects an action A_t and sends it to the environment. The environment's state changes to S_{t+1}. The agent receives an observation Z_{t+1} and a reward value R_t. Following this interaction description, the transition function T, observation function O, and reward function R are formally defined as follows.

- The transition function is defined as $T : S \times A \times S \rightarrow [0, 1]$. Given any state and action, s and a, the probability of the next possible state, s', is

  $$\mathcal{P}^a_{ss'} = T(s, a, s') = P\{S_{t+1} = s' | S_t = s, A_t = a\}, \text{ for all } t. \tag{2.1}$$

  These quantities are called *transition probabilities.* Transition function T is time-invariant and the sum of transition probabilities over the state space $\sum_{s' \in S} \mathcal{P}^a_{ss'} = 1, \text{ for all } (s, a)$.

- The observation function is defined as $O : S \times A \times Z \rightarrow [0, 1]$. Given any action and next state, a and s', the probability of the next observation, z', is

  $$\mathcal{P}^a_{s'z'} = O(s', a, z') = P\{Z_{t+1} = z' | A_t = a, S_{t+1} = s'\}, \text{ for all } t. \tag{2.2}$$

  These quantities are called *observation probabilities.* Observation function O is also time-invariant and the sum of observation probabilities over the observation space $\sum_{z' \in Z} \mathcal{P}^a_{s'z'} = 1, \text{ for all } (a, s')$.

- The reward function[3] is defined as $R : S \times A \to \mathbb{R}$. Given any current state and action, s and a, the *expected immediate reward* that the agent receives from the environment is
$$R_s^a = R(s, a) \tag{2.3}$$
Let R_{min} and R_{max} be the lower bound and upper bound of the reward function, that is to say
$$R_{min} < R(s, a) < R_{max}, \text{ for all } (s, a). \tag{2.4}$$

The state space might also contain some special *absorbing state* that only transitions to itself and the reward gained when the agent takes any action is 0. Suppose s is an absorbing state, we have

$$T(s, a, s') = \begin{cases} 1 \text{ if } s' = s \\ 0 \text{ otherwise} \end{cases} \text{ and } R(s, a) = 0, \text{ for all } a.$$

Given the POMDP model, we want to design a framework in which the agent's goal is to maximize the *expected cumulative reward* over time

$$V = E\left(\sum_{t=0}^{\mathcal{T}-1} \gamma^t R_t\right) \tag{2.5}$$

where $E(.)$ is the mathematical expectation, $\mathcal{T}$ is the *planning horizon* ($\mathcal{T} \geq 1$), γ is a *discount factor* ($0 \leq \gamma \leq 1$). The closer γ to 1 the more effect future rewards have on current agent action selection. This framework is called *finite-horizon* optimality. When $\mathcal{T} \to \infty$ and $\gamma < 1$, the framework is called *infinite-horizon* optimality. It is necessary to set the value of discount factor γ smaller than one in the infinite-horizon case to guarantee that the expected cumulative reward is bounded. From Equations 2.4 and 2.5, we have:

$$E\left(\sum_{t=0}^{\infty} \gamma^t R_{min}\right) < V < E\left(\sum_{t=0}^{\infty} \gamma^t R_{max}\right) \Rightarrow \frac{R_{min}}{1-\gamma} < V < \frac{R_{max}}{1-\gamma}. \tag{2.6}$$

2.5.2. Empathic dialogue agent example

To illustrate the main principle of a POMDP-based dialogue management, we use a simple empathic dialogue agent example for the tutoring domain, which is described as follows. A student ("the user") is interacting with the agent to learn a subject matter. The agent tries to infer the user's affective state to give an appropriate empathic feedback which aims to enhance the student's learning. Suppose that the user's affective state is either $s_1 = pos$ (positive) or $s_2 = neg$ (negative). The agent's goal is to select the most appropriate action from the following three actions: $a_1 =$

[3]We can also define the reward function as ($R : S \to \mathbb{R}$) or ($R : S \times A \times S \to \mathbb{R}$) or ($R : S \times A \times S \times Z \to \mathbb{R}$). However, the first definition is sufficient and does not change the fundamental properties of the framework.

comfort, $a_2 =$ *check*, and $a_3 =$ *encourage*. The *comfort* action is expressed by saying "I am sorry that you feel bad about the last question". The *check* action is the agent action to infer the user's affective state from outcome of an affect recognition module assumed to be available. The *encourage* action is expressed in the verbal form such as "Very good!" or "Well done!". If the agent knows exactly the user's true affective state, the action selection problem is trivial. The agent just selects the *encourage* action when the user's affective state is positive and the *comfort* action otherwise. Unfortunately, the user's affective state cannot be directly observed. Therefore, the agent must sometime use the *check* action to infer the user's affective state.

A POMDP model for this problem is represented by: (i) $S = \{s_1, s_2\} = \{pos, neg\}$, (ii) $A = \{a_1, a_2, a_3\} = \{comfort, check, encourage\}$, and (iii) $Z = \{z_1, z_2\} = \{\tilde{pos}, \tilde{neg}\}$. The transition function, observation function, and reward function are shown in Table 2.5. All transition and observation probabilities are handcrafted based on the common sense knowledge from the tutoring domain and affect recognition literature.

	$P(s'\|a,s)$			$P(z'\|a,s')$			$R(s,a)$	
a	s	$s' = pos$	$s' = neg$	s'	$z' = \tilde{pos}$	$z' = \tilde{neg}$	s	r
comfort	*pos*	0.80	0.20	*pos*	0.5	0.5	*pos*	-10
	neg	0.30	0.70	*neg*	0.5	0.5	*neg*	10
check	*pos*	0.90	0.10	*pos*	0.9	0.1	*pos*	-1
	neg	0.10	0.90	*neg*	0.1	0.9	*neg*	-1
encourage	*pos*	0.95	0.05	*pos*	0.5	0.5	*pos*	5
	neg	0.05	0.95	*neg*	0.5	0.5	*neg*	-10

Table 2.5: Transition function, observation function, and reward function for the empathic dialogue agent

The transition probability distribution in Table 2.5 is based on the following intuition. The user's affective state is *dynamic* and might change even without direct intervention from the agent. Therefore, when the agent selects the *check* action, which does not directly influence the user, the user's affective state might change from positive to negative or vice versa with probability 0.1. If the user is in a *pos* state, and the agent selects the *encourage* action which is the right one, the user, therefore, remains in a positive state with a high probability (0.95). Vice versa, if the user is in a *neg* state, the *encourage* is not appropriate and therefore, the chance that the negative state remains is high (0.95). Similarly, if the user's affective state is negative, the *comfort* action is appropriate and therefore the probability of changing to the positive state is higher than the *check* action.

When the agent selects the *check* action, it can infer the user's affective state with a 90% correctness (Table 2.5). The correctness rate here is interpreted as the classification accuracy of the affect recognition module as described in Section 2.4.1. When the agent selects *encourage* or *comfort* action, the observation probabilities are equally distributed.

Reward values are specified by the designer. They represent *what* the designer wants the agent to achieve, not *how* the designer wants it achieved [163, pg. 57].

In this example, we want the agent to select an appropriate action given uncertainty about the user's affective state. When the user's affective state is positive, the *encourage* action is appropriate and therefore receives a positive reward. When the user's state is negative, the *comfort* action is appropriate and a positive reward is assigned for this action. When the agent selects *check* action, it incurs a small negative reward and in return the agent is more certain about the user's current affective state.

2.5.3. Computing belief states

The state of the user cannot be directly observed. Therefore, in order to select good actions, the agent needs to maintain a complete trace of all the observations and actions that have happened so far. This trace is known as a *history*[4]. It is formally defined as:

$$H_{t+1} := \{A_0, Z_1, ..., Z_t, A_t, Z_{t+1}\}, \tag{2.7}$$

Astrom [10] showed that history H_{t+1} can be summarized via a *belief distribution.* A belief distribution is exactly the belief state of the agent.

$$B_{t+1}(s') = P\{S_{t+1} = s'|B_0, H_{t+1}\}, \tag{2.8}$$

Assuming the Markov property and using Bayes' rule, Equation 2.8 is transformed to the following equation (see the proof in *Smallwood and Sondik* [155], Appendix A):

$$B_{t+1}(s') = P\{S_{t+1} = s'|Z_{t+1} = z', Z_t = a, B_t = b\} \tag{2.9}$$

Formally, let the belief space B be an infinite set of belief states. A belief state $b \in B$ is encoded as a $|S|$-dimensional column vector $(b_1, b_2, ..., b_{|S|})^T$, where each element $b_i = b(s_i)$ is the probability that the current state of the environment is s_i, $b_i \geq 0, \forall i \in [1, |S|]$, and $\sum_{i=1}^{|S|} b_i = 1$. Geometrically, a belief state is a point in a $(|S| - 1)$-dimensional belief simplex.

Concretely, the agent starts with an initial belief state $B_0 = b_0$. At time t, the agent's belief is $B_t = b$, it selects action $A_t = a$ and sends this to the environment. The state changes to $S_{t+1} = s'$. State S_{t+1} cannot be directly observed and the agent only gets observation $Z_{t+1} = z$. The agent also receives a reward $R_t = r$, the value of which depends on the actual values of state s and agent's action a. At this moment the agent needs to update its belief state $B_{t+1} = b'$ given known values for b, a, z. Starting from Equation 2.9, $b'(s')$ is computed using the basic laws from the probability theory as follows:

[4]In a dialogue management context, this trace is the dialogue history

$$\begin{aligned}
b'(s') &= P(s'|z,a,b) \\
&= \frac{P(z|s',a,b)P(s'|a,b)}{P(z|a,b)} \\
&= \frac{P(z|s',a)P(s'|a,b)}{P(z|a,b)}, \text{ } (Z_{t+1} \text{ and } B_t \text{ are independent}) \\
&= \frac{\mathcal{P}^a_{s'z}\sum_{s\in S}P(s'|a,b,s)P(s|a,b)}{P(z|a,b)} \\
&= \frac{\mathcal{P}^a_{s'z}\sum_{s\in S}P(s'|a,s)P(s|b)}{P(z|a,b)} \\
&= \frac{\mathcal{P}^a_{s'z}\sum_{s\in S}\mathcal{P}^a_{ss'}b(s)}{P(z|a,b)} \\
&= \frac{\mathcal{P}^a_{s'z}T^a_{s'}b}{P(z|a,b)}, \\
&= \eta\mathcal{P}^a_{s'z}T^a_{s'}b,
\end{aligned} \tag{2.10}$$

where $T^a_{s'}$ is a $|S|$-dimensional row vector:

$$T^a_{s'} = \left(\mathcal{P}^a_{s_1s'}, ..., \mathcal{P}^a_{s_{|S|}s'}\right), |S| \text{ is the number of elements of } S.$$

$\eta = 1/P(z|a,b)$ is a normalizing constant, independent of state s'.

The belief state b' is represented as

$$b' = \eta W^a_z b \tag{2.11}$$

where W^a_z is a $|S| \times |S|$ matrix,

$$W^a_z = \begin{bmatrix} \mathcal{P}^a_{s_1z}\mathcal{P}^a_{s_1s_1} & \mathcal{P}^a_{s_1z}\mathcal{P}^a_{s_2s_1} & ... & \mathcal{P}^a_{s_1z}\mathcal{P}^a_{s_{|S|}s_1} \\ \mathcal{P}^a_{s_2z}\mathcal{P}^a_{s_1s_2} & \mathcal{P}^a_{s_2z}\mathcal{P}^a_{s_2s_2} & ... & \mathcal{P}^a_{s_2z}\mathcal{P}^a_{s_{|S|}s_2} \\ ... & ... & ... & ... \\ \mathcal{P}^a_{s_{|S|}z}\mathcal{P}^a_{s_1s_{|S|}} & \mathcal{P}^a_{s_{|S|}z}\mathcal{P}^a_{s_2s_{|S|}} & ... & \mathcal{P}^a_{s_{|S|}z}\mathcal{P}^a_{s_{|S|}s_{|S|}} \end{bmatrix} \tag{2.12}$$

The agent-environment interaction process for planning horizon $\mathcal{T}$ is shown in Figure 2.11a. When $\mathcal{T} > 1$, the interaction process can be represented as a DBN as in Figure 2.11b.

Let us now illustrate the belief monitoring process of the empathic dialogue agent example where the agent selects an action randomly (e.g. at the beginning of the learning process). At the beginning, the agent starts with a uniform belief state $B_0 = (0.5\ 0.5)^T$. The agent selects a random action, let say $A_0 = check$ and receives an immediate reward $R_0 = -1$ and an observation $Z_1 = \tilde{pos}$. The belief state B_1 is computed as follows:

$$B_1 = \eta W^{A_0}_{Z_1} B_0$$

From Equation 2.12 and Table 2.5, we have:

$$W^{A_0}_{Z_1} = \begin{bmatrix} \mathcal{P}^{A_0}_{s_1Z_1}\mathcal{P}^{A_0}_{s_1s_1} & \mathcal{P}^{A_0}_{s_1Z_1}\mathcal{P}^{A_0}_{s_2s_1} \\ \mathcal{P}^{A_0}_{s_2Z_1}\mathcal{P}^{A_0}_{s_1s_2} & \mathcal{P}^{A_0}_{s_2Z_1}\mathcal{P}^{A_0}_{s_2s_2} \end{bmatrix} = \begin{bmatrix} 0.9 \times 0.9 & 0.9 \times 0.1 \\ 0.1 \times 0.1 & 0.1 \times 0.9 \end{bmatrix} = \begin{bmatrix} 0.81 & 0.09 \\ 0.01 & 0.09 \end{bmatrix}$$

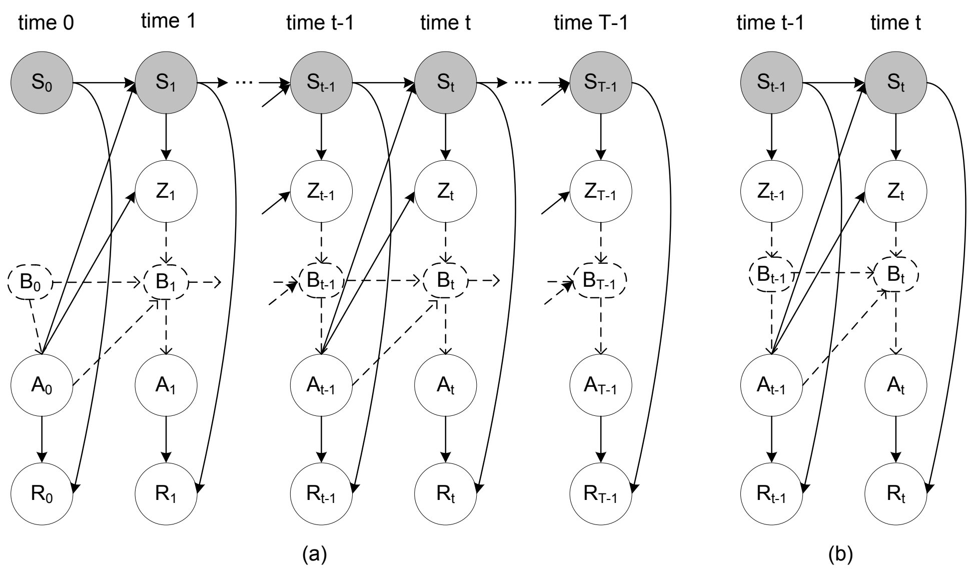


Figure 2.11: (a) Bayesian network representation illustrating the agent reasoning process in $\mathcal{T}$ steps, (b) Equivalent dynamic Bayesian network representation ($\mathcal{T} > 1$). The shaded nodes are hidden, the clear nodes are observable. The dashed nodes (i.e. belief nodes) are shown to clarify how the system updates its internal beliefs and selects actions. In the actual implementation, these nodes are derived from the hidden nodes.

Substitute the computed value $W_{Z_1}^{A_0}$ to Equation 2.5.3 we have:

$$B_1 = \eta \begin{bmatrix} 0.81\ 0.09 \\ 0.01\ 0.09 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \eta \begin{bmatrix} 0.45 \\ 0.05 \end{bmatrix}$$

Normalize B_1 so that $B_1(s_1) + B_1(s_2) = 1$ we have $B_1 = (0.9\ 0.1)^T$. Note that we do not need to compute η.

Now starting with B_1, the agent selects a random action, let say $A_1 = \textit{encourage}$. It receives an immediate reward $R_1 = 5$ and an observation $Z_2 = \tilde{pos}$. Similarly, we can easily compute B_2 from B_1, A_1, Z_2. It will result in:

$$B_2 = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}$$

2.5.4. Finding an optimal policy

A policy is a function:

$$\pi(b) \longrightarrow a, \tag{2.13}$$

where b is a belief state and a is the action chosen by the policy π.

An optimal policy π^* is a policy that maximizes the expected cumulative reward:

$$\pi^* = argmax_\pi E\left[\sum_{t=0}^{\infty} \gamma^t R_t\right], \tag{2.14}$$

where R_t is the reward when the agent follows policy π.

We define value functions $V_i : B \rightarrow \mathbb{R}$. $V_n(b)$ is the maximum expected cumulative reward when the agent has n remaining steps to go. Its associated policy is denoted by π_n. When the agent has only one remaining step to go (i.e. $n = 1$), all it can do is to select an action and send it to the environment, we have:

$$\begin{aligned} V_1(b) &= \max_{a \in A} \sum_{s \in S} R(s,a) b(s) \\ &= \max_{a \in A} r_a b, \end{aligned} \tag{2.15}$$

where r_a is a row vector, $r_a = \left(R_{s_1}^a, ..., R_{s_{|S|}}^a\right)$.

When the agent has n remaining steps to go ($n > 1$), the value function V_n is defined inductively as [155]:

$$V_n(b) = \max_{a \in A}\left[r_a b + \gamma \sum_{z \in Z} P(z|a,b) V_{n-1}(b_a^z)\right] \tag{2.16}$$

where b_a^z is the belief state of the agent after selecting action a, and the observation of the environment change to z.

When $n \to \infty$, the optimal value function for the infinite-horizon case is denoted by V^*. Puterman[131, Theorem 6.9] proved that V_n converges to V^* when n goes to infinity. Therefore, from Equation 2.16 we have:

$$V^*(b) = \max_{a \in A} \left[r_a b + \gamma \sum_{z \in Z} P(z|a,b) V^*(b_a^z) \right] \tag{2.17}$$

For any positive number ϵ, the policy π_n is *ϵ-optimal* if

$$V^*(b) - V_n(b) \leq \epsilon \text{ for all } b \in B. \tag{2.18}$$

Equation 2.16 is used to develop an important type of algorithms called Value Iteration (VI) (Section 2.6). Value iteration is an algorithm for finding *ϵ-optimal* policies. It is terminated when [131]:

$$\sup_b |V_n(b) - V_{n-1}(b)| \leq \frac{\epsilon(1-\gamma)}{2\gamma}, \tag{2.19}$$

where $\sup|X|$ stands for supremum norm of set X. The left part of Equation 2.19 is called the *Bellman residual.*

Because there are an infinite number of belief states, we cannot compute V_{n-1} directly for each belief state b. *Sondik* [158] proved that V_{n-1} can be represented through a finite set of α-vectors $\Gamma_{n-1} = \{\alpha_1, ..., \alpha_{|\Gamma_{n-1}|}\}$, where each vector $\alpha \in \Gamma_{n-1}$ is a $|S|$-dimensional row vector (also called a *hyperplane*, hereafter it is called an *α-vector*), and

$$V_{n-1}(b) = \max_{\alpha \in \Gamma_{n-1}} \alpha b \tag{2.20}$$

Therefore, from Equations 2.11 and 2.20 we can rewrite Equation 2.16 as

$$\begin{aligned}
V_n(b) &= \max_{a \in A} \left[r_a b + \gamma \sum_{z \in Z} P(z|a,b) \max_{\alpha \in \Gamma_{n-1}} \alpha b_a^z \right] \\
&= \max_{a \in A} \left[r_a b + \gamma \sum_{z \in Z} P(z|a,b) \max_{\alpha \in \Gamma_{n-1}} \alpha \frac{W_z^a b}{P(z|a,b)} \right] \\
&= \max_{a \in A} \left[r_a b + \gamma \sum_{z \in Z} \max_{\alpha \in \Gamma_{n-1}} \alpha W_z^a b \right] \\
&= \max_{a \in A} \left[r_a b + \gamma (\max_{l_1} \alpha_{l_1} . W_{z_1}^a b + ... + \max_{l_{|Z|}} \alpha_{l_{|Z|}} W_{z_{|Z|}}^a b) \right] \\
&= \max_{a \in A} \left[r_a b + \gamma \max_{l_1} ... \max_{l_{|Z|}} (\alpha_{l_1} W_{z_1}^a b + ... + \alpha_{l_{|Z|}} W_{z_{|Z|}}^a b) \right] \\
&= \max_{a \in A} \left[r_a b + \gamma \max_{l_1} ... \max_{l_{|Z|}} \sum_{k=1}^{|Z|} \alpha_{l_k} W_{z_k}^a b \right] \\
&= \max_{a \in A} \max_{l_1} ... \max_{l_{|Z|}} \left[r_a + \gamma \sum_{k=1}^{|Z|} \alpha_{l_k} W_{z_k}^a \right] b,
\end{aligned} \tag{2.21}$$

where $l_1, l_2, ..., l_{|Z|} \in [1, |\Gamma_{n-1}|]$.

The set Γ_n can now be generated from set Γ_{n-1} by the following update:

$$\Gamma_n \leftarrow \alpha' = r_a + \gamma \sum_{k=1}^{|Z|} \alpha_{l_k}.W_k^a, \forall a \in A, \alpha_{l_k} \in \Gamma_{n-1}, \tag{2.22}$$

where $n \geq 1$ and $\Gamma_1 = \{r_{a_1}, r_{a_2}, ..., r_{a_{|A|}}\}$.

Finding the optimal policy (for planning horizon $\mathcal{T} = n$) is now considered as solving a set of $|A||\Gamma_{n-1}|^{|Z|}$ linear constraints derived from Equation 2.21. To gain the computational tractability, it is necessary to keep only the vectors that contribute to the optimal value function because the number of α-vectors generated from Equation 2.22 is very large. We distinguish two types of α-vectors: *useful* vectors and *extraneous* vectors [190]. A vector $\alpha \in \Gamma_n$ is useful[5] if:

$$\exists b \in B : \alpha b > \alpha' b, \text{for all } \alpha' \in \Gamma_n - \alpha \tag{2.23}$$

A vector $\alpha' \in \Gamma_n$ that does not satisfy Equation 2.23 is an extraneous vector. A set Γ_n that is composed of useful vectors is called a *parsimonious* set [188]. From Equation 2.20 it is obvious that we can safely remove all the extraneous vectors from the set Γ_n. *Monahan* [112] proposed a procedure to remove extraneous vectors by solving the following linear program for each $\alpha \in \Gamma_n$:

$$\begin{aligned} &\textit{variables: } x, b_i, \forall i \in [1, |S|] \\ &\textit{maximize } x \\ &\textit{subject to constraints: } b(\alpha - \alpha') \geq x; \forall \alpha' \in \Gamma_n \;\&\; \sum_{i=1}^{|S|} b_i = 1 \end{aligned} \tag{2.24}$$

If $x < 0$, remove α from Γ_n.

When the set of useful α-vectors Γ_n is found. The agent's action $\hat{a}$ is determined as $\hat{a} \leftarrow \hat{\alpha}$[6], where

$$\hat{\alpha} = \operatorname*{argmax}_{\alpha \in \Gamma_n} \alpha b \tag{2.25}$$

Let us again illustrate the process of finding the optimal policy for the empathic dialogue agent example with the discount factor $\gamma = 0.95$. For $n = 1$, we have

$$\Gamma_1 = \left\{ \begin{matrix} \alpha_1, \\ \alpha_2, \\ \alpha_3 \end{matrix} \right\} = \left\{ \begin{matrix} r_{a_1}, \\ r_{a_2}, \\ r_{a_3} \end{matrix} \right\} = \left\{ \begin{matrix} \text{(-10 10)}, \\ \text{(-1 -1)}, \\ \text{(5 -10)} \end{matrix} \right\} \tag{2.26}$$

All vectors of the set Γ_1 are useful vectors (see Fig. 2.12a). Apply Equation 2.20, we

[5]Assume that the identical vectors in Γ_n are merged.

[6]Because each α-vector is associated with only one action, see Equation 2.22

have

$$V_1(b) = \max_{\alpha \in \Gamma_0} \alpha b$$
$$= \begin{cases} -10b_1 + 10b_2 & \text{if } 0.00 \leq b_1 \leq 0.55, \\ -1 & \text{if } 0.55 < b_1 \leq 0.60 \\ 5b_1 - 10b_2 & \text{if } 0.60 < b_1 \leq 1.00, \end{cases} \tag{2.27}$$

where $b_2 = 1 - b_1$.

For $n = 2$, Γ_2 is composed of $|A||\Gamma_1|^{|Z|} = 27$ vectors generated from Γ_1 as follows:

$$\Gamma_2 = \left\{ \begin{array}{r} \alpha'_1, \\ \alpha'_2, \\ \alpha'_3, \\ \alpha'_4, \\ \alpha'_5, \\ \alpha'_6, \\ \alpha'_7, \\ \alpha'_8, \\ \alpha'_9, \\ \alpha'_{10}, \\ \alpha'_{11}, \\ \alpha'_{12}, \\ \alpha'_{13}, \\ \alpha'_{14}, \\ \alpha'_{15}, \\ \alpha'_{16}, \\ \alpha'_{17}, \\ \alpha'_{18}, \\ \alpha'_{19}, \\ \alpha'_{20}, \\ \alpha'_{21}, \\ \alpha'_{22}, \\ \alpha'_{23}, \\ \alpha'_{24}, \\ \alpha'_{25}, \\ \alpha'_{26}, \\ \alpha'_{27} \end{array} \right\} = \left\{ \begin{array}{c} r_{a_1} + \gamma\alpha_1 W^{a_1}_{z_1} + \gamma\alpha_1 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_1 W^{a_1}_{z_1} + \gamma\alpha_2 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_1 W^{a_1}_{z_1} + \gamma\alpha_3 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_2 W^{a_1}_{z_1} + \gamma\alpha_1 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_2 W^{a_1}_{z_1} + \gamma\alpha_2 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_2 W^{a_1}_{z_1} + \gamma\alpha_3 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_3 W^{a_1}_{z_1} + \gamma\alpha_1 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_3 W^{a_1}_{z_1} + \gamma\alpha_2 W^{a_1}_{z_2}, \\ r_{a_1} + \gamma\alpha_3 W^{a_1}_{z_1} + \gamma\alpha_3 W^{a_1}_{z_2}, \\ r_{a_2} + \gamma\alpha_1 W^{a_2}_{z_1} + \gamma\alpha_1 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_1 W^{a_2}_{z_1} + \gamma\alpha_2 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_1 W^{a_2}_{z_1} + \gamma\alpha_3 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_2 W^{a_2}_{z_1} + \gamma\alpha_1 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_2 W^{a_2}_{z_1} + \gamma\alpha_2 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_2 W^{a_2}_{z_1} + \gamma\alpha_3 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_3 W^{a_2}_{z_1} + \gamma\alpha_1 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_3 W^{a_2}_{z_1} + \gamma\alpha_2 W^{a_2}_{z_2}, \\ r_{a_2} + \gamma\alpha_3 W^{a_2}_{z_1} + \gamma\alpha_3 W^{a_2}_{z_2}, \\ r_{a_3} + \gamma\alpha_1 W^{a_3}_{z_1} + \gamma\alpha_1 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_1 W^{a_3}_{z_1} + \gamma\alpha_2 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_1 W^{a_3}_{z_1} + \gamma\alpha_3 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_2 W^{a_3}_{z_1} + \gamma\alpha_1 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_2 W^{a_3}_{z_1} + \gamma\alpha_2 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_2 W^{a_3}_{z_1} + \gamma\alpha_3 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_3 W^{a_3}_{z_1} + \gamma\alpha_1 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_3 W^{a_3}_{z_1} + \gamma\alpha_2 W^{a_3}_{z_2}, \\ r_{a_3} + \gamma\alpha_3 W^{a_3}_{z_1} + \gamma\alpha_3 W^{a_3}_{z_2} \end{array} \right\} = \left\{ \begin{array}{r} \text{(-15.7 13.8)}, \\ \text{(-13.325 11.425)}, \\ \text{(-13.325 11.425)}, \\ \text{(-11.9 9.2875)}, \\ \text{(-11.9 9.2875)}, \\ \text{(-10.95 9.05)}, \\ \text{(-9.525 6.9125)}, \\ \text{(-9.525 6.9125)}, \\ \text{(-9.0275 -8.6475)}, \\ \text{(-8.771 -1.779)}, \\ \text{(-8.6 6.6)}, \\ \text{(-8.1 4.775)}, \\ \text{(-3.55 -1.45)}, \\ \text{(-2.2065 -8.8185)}, \\ \text{(-1.95 -1.95)}, \\ \text{(-1.779 6.429)}, \\ \text{(0.25 -6.2)}, \\ \text{(0.25 -6.2)}, \\ \text{(2.325 -9.075)}, \\ \text{(2.5815 -2.2065)}, \\ \text{(2.74375 -10.11875)}, \\ \text{(2.74375 -10.11875)}, \\ \text{(2.7525 6.1725)}, \\ \text{(4.05 -10.95)}, \\ \text{(6.54375 -14.86875)}, \\ \text{(6.54375 -14.86875)}, \\ \text{(9.0375 -18.7875)} \end{array} \right\} \tag{2.28}$$

Six pairs of α-vectors of Γ_2 are identical: $\{\alpha'_2, \alpha'_3\}$, $\{\alpha'_4, \alpha'_5\}$, $\{\alpha'_7, \alpha'_8\}$, $\{\alpha'_{17}, \alpha'_{18}\}$, $\{\alpha'_{21}, \alpha'_{22}\}$, and $\{\alpha'_{25}, \alpha'_{26}\}$. After merging these pairs, we have the set of 21 nonidentical vectors Γ_2'. These vectors are represented in Figure 2.12b. Among these 21 vectors, there are only three useful vectors: α_1', α_{23}', and α_{27}'. Pruning the extraneous vectors from the set Γ_2' we have a new set of useful vectors $\Gamma_2'' = \{\alpha_1', \alpha_{23}', \alpha_{27}'\}$.

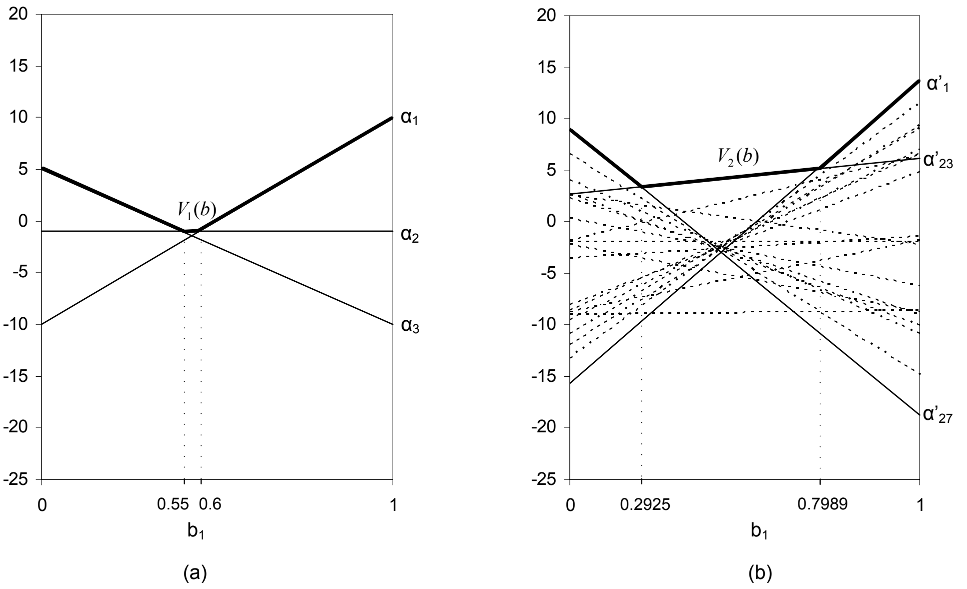


Figure 2.12: Representation of α-vectors of (a) Γ_1 and (b) Γ_2'. Solid lines are useful vectors. Dashed lines are extraneous vectors. Upper bounds of the solid lines (i.e. bold lines) are optimal value functions.

We now compute the optimal value function $V_2(b)$ as the following.

$$V_2(b) = \max_{\alpha \in \Gamma_2''} \alpha b$$
$$= \begin{cases} -15.7b_1 + 13.8b_2 & \text{if } 0.0000 \le b_1 \le 0.2925 \\ 2.7525b_1 + 6.1725b_2 & \text{if } 0.2925 < b_1 \le 0.7989 \\ 9.0375b_1 - 18.7875b_2 & \text{if } 0.7989 < b_1 \le 1.0000, \end{cases} \tag{2.29}$$

where $b_2 = 1 - b_1$.

Note that each α-vector is associated with an action (as we can see from Equations 2.26 and 2.28). Therefore, the action selection problem is equivalent to selecting an α-vector from a set of useful vectors given the agent's belief state. For example, given the initial belief state $B_0 = \{0.5\ 0.5\}$, the best action the agent can choose at the beginning based on V_1 is $a_1 =$ *comfort* (correspond to α_1) and based on $V_2(b)$ is $a_3 =$ *encourage* (correspond to α_{23}').

Continuing this process, we obtained two distinctive properties of this problem. First, Figure 2.13 shows that the number of useful α-vectors does not increase exponentially with the number of planning horizons. Indeed, the number of useful α-vectors is a constant (equal to 13) when $\mathcal{T} \geq 26$.

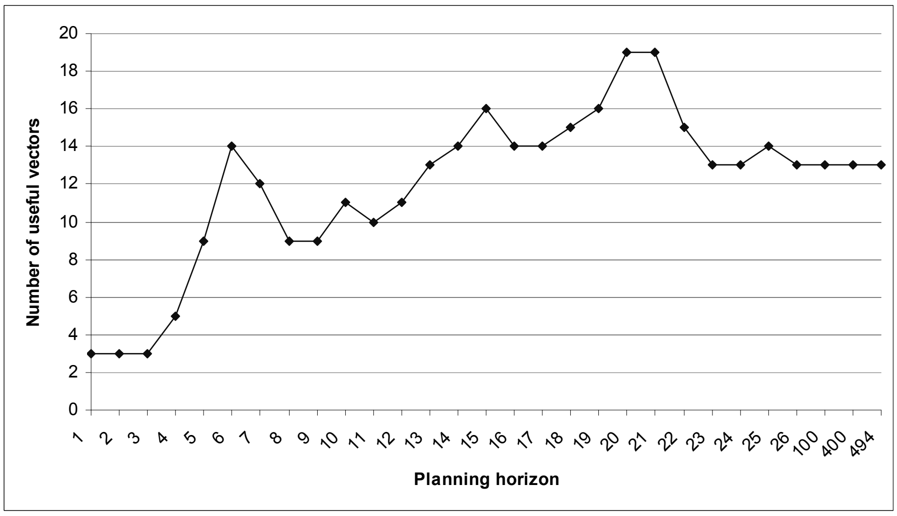


Figure 2.13: Number of useful vectors vs. planning horizon

Second, the optimal value function converges at $\mathcal{T} = 494$ ($\epsilon = 10^{-9}$). This means we also obtain the optimal policy for the infinite-horizon case as well when $\mathcal{T} = 494$. The optimal value functions for $\mathcal{T} = 10$, $\mathcal{T} = 20$, and $\mathcal{T} = 494$ are shown in Figure 2.14.

As mentioned, the optimal value function for the infinite-horizon case $V^*(b)$ is equal to $V_{494}(b)$ and is computed as follows:

$$V^*(b) \approx V_{494}(b) = \begin{cases} 28.4278b_1 + 58.6604b_2 & \text{if } 0.0000 \leq b_1 \leq 0.1004 \\ 35.3949b_1 + 57.8826b_2 & \text{if } 0.1004 < b_1 \leq 0.2098 \\ 36.1418b_1 + 57.6843b_2 & \text{if } 0.2098 < b_1 \leq 0.3502 \\ 46.7367b_1 + 51.9739b_2 & \text{if } 0.3502 < b_1 \leq 0.4049 \\ 47.9208b_1 + 51.1681b_2 & \text{if } 0.4049 < b_1 \leq 0.5015 \\ 48.4500b_1 + 50.6358b_2 & \text{if } 0.5015 < b_1 \leq 0.5355 \\ 49.1138b_1 + 49.8703b_2 & \text{if } 0.5355 < b_1 \leq 0.7548 \\ 49.3500b_1 + 49.1432b_2 & \text{if } 0.7548 < b_1 \leq 0.7563 \\ 49.3969b_1 + 48.9976b_2 & \text{if } 0.7563 < b_1 \leq 0.8319 \\ 51.9081b_1 + 36.5667b_2 & \text{if } 0.8319 < b_1 \leq 0.8688 \\ 53.5840b_1 + 25.4671b_2 & \text{if } 0.8688 < b_1 \leq 0.9098 \\ 54.5692b_1 + 15.5293b_2 & \text{if } 0.9098 < b_1 \leq 0.9553 \\ 54.9864b_1 + 6.60722b_2 & \text{if } 0.9553 < b_1 \leq 1.0000, \end{cases} \tag{2.30}$$

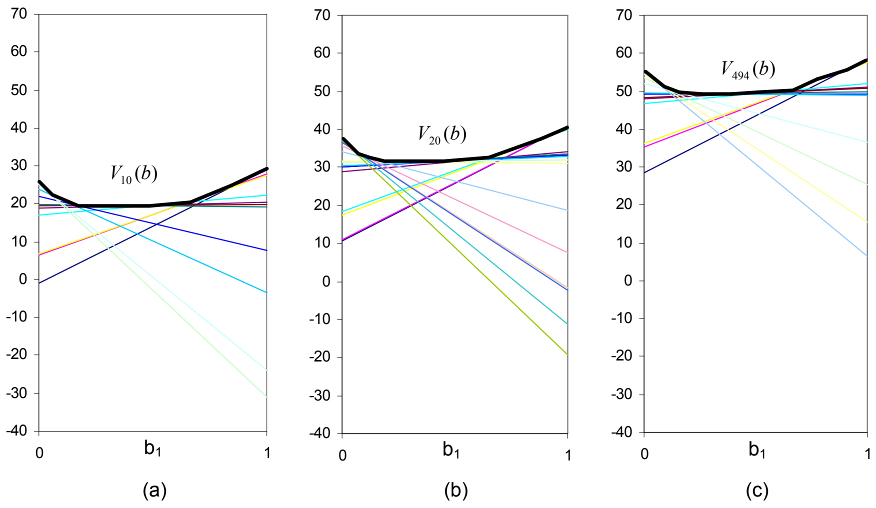


Figure 2.14: Representation of the optimal value functions for different planning horizons (a) $\mathcal{T} = 10$, (b) $\mathcal{T} = 20$, and (c) $\mathcal{T} = 494$. Lines are useful vectors. Upper bounds of these lines (i.e. bold lines) are optimal value functions.

where $b = (b_1, b_2)^T$ and $b_2 = 1 - b_1$.

The optimal policy can also be represented by a policy graph (Fig. 2.15). A policy graph is a directed graph where each of the nodes are the agent's actions and the arcs are the observations. Each node in the policy graph is associated with an α-vector in the set Γ. The index i below the action label of each node corresponds to the α-vector i^{th} in Γ. For example, each index of the policy graph in Figure 2.15 corresponds to one vector in Equation 2.30. An advantage of using the policy graph as an action selection function is that the agent does not need to compute its belief state during the system-user interaction process as described at the beginning of this section.

2.6. Review of POMDP solution techniques

In this section, we present a popular class of POMDP algorithms, called Value Iteration (VI), that are usually used to compute an optimal or near-optimal policy (based on the literature). Other POMDP algorithms are reviewed in *Aberdeen* [1], *Murphy* [113], *Thrun et al.* [164]. Section 2.6.1 is about the exact VI algorithms. Section 2.6.2 describes approximate Point-Based Value Iteration (PBVI) algorithms which are used to compute near-optimal policies for dialogue problems presented in Chapters 5 and 6.

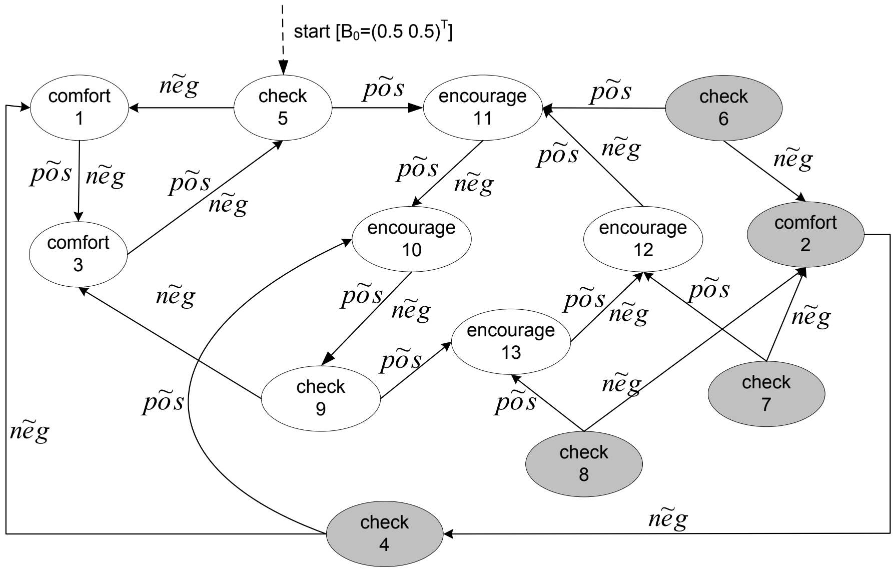


Figure 2.15: Optimal policy for the empathic dialogue agent represented as a policy graph. Given the initial belief $B_0 = (0.5, 0.5)^T$ the start node is $check_5$. Shaded nodes are unreachable from $check_5$, therefore we can remove them from the policy graph.

2.6.1. Exact value iteration algorithms

In this section, we present three representative algorithms that aim to illustrate core ideas of the exact VI solution. Algorithm 1 (derived from Eq. 2.16 and 2.19) is a general form (i.e., global level) of all exact VI algorithms. Figure 2.16 shows a conceptual idea of how to compute the Bellman residual r (Alg. 1) for two sets of alpha-vectors $\Gamma = \{\alpha_1, \alpha_2, \alpha_3\}$ and $\Gamma' = \{\alpha'_1, \alpha'_2\}$. Further technical details about computing the Bellman residual is presented in *Zhang and Zhang* [190]. Algorithm 2, the simplest algorithm, is used to show the computational complexity issue. Algorithm 3, pruning extraneous vectors, is an important routine used in many recent algorithms (including Alg. 2).

Almost all exact value iteration algorithms differ only in the way they compute the set of useful vectors Γ_n given the set of useful vectors Γ_{n-1}, which is represented as the *update*(Γ) function in Algorithm 1. In the following, we discuss them in detail.

Algorithm 2, called *Enumeration*, is derived from Equations 2.22 and 2.24 in Section 2.5.4. It was proposed by *Sondik* [158] and *Monahan* [112]. The main idea of the enumeration algorithm is to use the dynamic programming technique to generate the set of all α-vectors Γ_n from Γ_{n-1} and to prune them to get the set of useful α-vectors using linear programming. This algorithm is inefficient because the step to generate the set Γ_n from Γ_{n-1} might be computationally expensive ($|\Gamma_n| = |A||\Gamma_{n-1}|^{|Z|}$).

Algorithm 1: exactVI($\Gamma_0, \gamma, \epsilon$) (Value iteration for POMDPs)

```
Input: Initial set of vectors $\Gamma_0$, discount factor $\gamma$, and positive number $\epsilon$
Output: Set of useful vectors $\Gamma$
begin
    $\Gamma \leftarrow \Gamma_0$; $\delta \leftarrow \frac{\epsilon(1-\gamma)}{2\gamma}$; $r \leftarrow \infty$;
    while $r > \delta$ do
        $\Gamma' \leftarrow$ update($\Gamma$);
        $r \leftarrow \max_{b \in B, \alpha \in \Gamma, \alpha' \in \Gamma'} |\alpha' b - \alpha b|$; // See [190] for a proposal to
        compute the Bellman residual.
        $\Gamma \leftarrow \Gamma'$;
    return $\Gamma$;
end
```

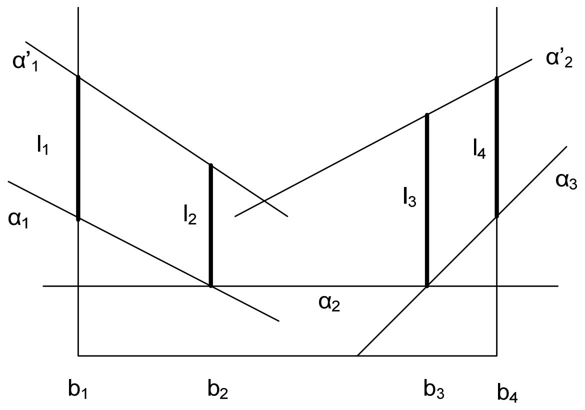


Figure 2.16: Conceptual idea of how to compute the Bellman residual r in Algorithm 1: $r = max\{l_1, l_2, l_3, l_4\}$, where $\Gamma = \{\alpha_1, \alpha_2, \alpha_3\}$ and $\Gamma' = \{\alpha'_1, \alpha'_2\}$.

For example, the run-time to solve the 4×3 maze problem [120] is more than eight hours([41], see Table 2.6). This is because, unlike the emphatic dialogue agent problem, the number of useful α-vectors of the 4×3 maze problem increases quickly in the planning horizon. The numbers of useful vectors when the planning horizon increases from 1 to 10 are: $1, 3, 4, 4, 15, 41, 133, 430, 1289, 3593$. In consequence, at $\mathcal{T} = 11$, there is already a very large number of α-vectors (4×3593^6) being generated from the set of useful vectors Γ_{10}.

```
Algorithm 2: update($\Gamma_{n-1}$) (Enumeration [112, 158])
Input: Set of useful vectors $\Gamma_{n-1}$
Output: Set of useful vectors $\Gamma_n$
begin
    $\Gamma_n \leftarrow \emptyset$;
    forall actions $a \in A$ do
        forall $(l_1, l_2, ..., l_{|Z|}) = (1, 1, ..., 1)$ to $(|\Gamma_{n-1}|, |\Gamma_{n-1}|, ..., |\Gamma_{n-1}|)$ do
            $\alpha' \leftarrow \left[ r_a + \gamma \sum_{k=1}^{|Z|} \alpha_{l_k} W_{z_k}^a \right]$;
            $\Gamma_n \leftarrow \Gamma_n \bigcup (a; \alpha')$;
    return $\Gamma_n \leftarrow$ prune($\Gamma_n$); // See Alg. 3
end
```

To alleviate this computational difficulty, *Sondik* [158] proposed another algorithm, called *one-pass*, that can generate the set of useful α-vectors Γ_n directly. Instead of generating all α-vectors, the one-pass algorithm only produces a set of useful vectors. The main idea of the algorithm is explained as follows. It starts with an initial belief state b. It then finds the useful α-vector for b and the region in which this vector is dominant. After that, it continues with the start and end points of this region to find other useful α-vectors. The process is then repeated until no new useful α-vector is found.

Cheng [44] starts with Sondik's one-pass idea but opts for less strict constraints. The linear support algorithm starts with extreme points of the belief space. It then picks a point, generates the vector for that point, and checks the region of this vector to see if it is the correct one at all corners (vertices) of the region.

Instead of computing the set of useful vectors Γ_n directly, *Cassandra et al.* [40] first convert the value function V_n to the state-action value function Q_n^a for each action a:

$$Q_n^a(b) = r_a b + \gamma \sum_{z \in Z} P(z|a,b) V_{n-1}(b_a^z). \tag{2.31}$$

Then they compute a set of useful vectors Γ_n^a for $Q_n^a(b)$ and prune the extraneous vectors. See *Kaelbling et al.* [89] for further details of the algorithm. They also show that Witness performs better than the previous algorithms for a class of problems which have a small number of useful α-vectors Γ_n^a.

Zhang and Liu [189] use a clever pruning technique to further reduce the computational intractability. Their algorithm, Incremental Pruning(IP), is the most efficient

Algorithm 3: prune(Γ) (Adapted from [62])

```
Input: Set of vectors $\Gamma$
Output: Set of useful vectors $\Gamma'$
begin
    $\Gamma' \leftarrow \emptyset$;
    while $\Gamma \neq \emptyset$ do
        $\alpha \leftarrow$ any element in $\Gamma$;
        if pointwise-dominate($\alpha, \Gamma$) then $\Gamma = \Gamma - \{\alpha\}$;
        else
            $b \leftarrow$ lp-dominate($\alpha, \Gamma$);
            if $b = nil$ then $\Gamma = \Gamma - \alpha$ else
                $\alpha \leftarrow$ best($b, \Gamma$);
                $\Gamma' \leftarrow \Gamma' \bigcup \{\alpha\}$;
                $\Gamma \leftarrow \Gamma - \alpha$;
end
pointwise-dominate($\alpha, \Gamma$):
begin
    forall $\alpha' \in \Gamma$ do
        if $\alpha_i \leq \alpha'_i, \forall i \in [1, |S|]$ then return true;
    return false;
end
lp-dominate($\alpha, \Gamma$):
begin
    solve the following linear programs:
    variables: $x, b_i, \forall i \in [1, |S|]$
    maximize: $x$
    subject to the constraints: $b(\alpha - \alpha') \geq x, \forall \alpha' \in \Gamma$ & $\sum_{i=1}^{|S|} b_i = 1$
    if $x \geq 0$ then return $b$ else return $nil$;
end
best($b, \Gamma$):
begin
    $max \leftarrow -\infty$
    forall $\alpha \in \Gamma$ do
        if $(\alpha b > max)$ or $((\alpha b = max)$ and $(\alpha <_{lex} \alpha'))$ then
            $\alpha' \leftarrow \alpha$; // $<_{lex}$ denotes lexicographic ordering (see [39,
            pg. 67])
            $max \leftarrow \alpha b$;
    return $\alpha'$
end
```

exact algorithm when compared with previous algorithms. Recent work reported a number of extensions from the IP algorithm: GIP [41], RR [41], IBIP [62], and RBIP [63].

Table 2.6 [41] shows the benchmark of some of these exact POMDP algorithms[7]. We can see that all these algorithms can only handle problems with a few dozen states, actions, and observations. As illustrated in Section 2.5.4, the optimal policy is computed for all belief points in the belief simplex B, many of them are unreachable from the initial belief state B_0. In consequence, the complexity of the exact algorithms grows exponentially with the planning horizon $\mathcal{T}$ (curse of history problem). They, therefore, are not appropriate for solving realistic dialogue management problems except toy problems such as the empathic agent example presented in Section 2.5.2. Fortunately, recent progress in finding approximate algorithms provides some hope for designing more realistic POMDP-based dialogue management models. We will discuss these algorithms in the next section.

Specification				Time (s)		
Problem	$\|S\|$	$\|A\|$	$\|Z\|$	Enumeration	Witness	Incremental Pruning
1D maze	4	2	2	2.2	9.3	2.3
Part painting	4	4	2	1116.9	5608.4	4249.2
Network	7	4	2	>8h	6422.9	1066.6
Shuttle	8	3	5	>8h	1676.7	200.8
4x3 maze	11	4	6	>8h	727.1	346.0
Cheese	11	4	7	>8h	351.8	215.7
4x3 CO	11	4	11	>8h	24.6	22.8
Aircraft ID	12	6	5	>8h	417.0	234.1
4x4 maze	16	4	2	216.7	3226.0	1557.0

Table 2.6: Benchmark of exact POMDP algorithms for small problems (computed on a Sun SPARC-10 computer) [41]

2.6.2. Approximate value iteration algorithms

The main idea of an approximate point-based VI algorithm is to approximate the entire belief space by a set of reachable belief points B_r starting from the initial belief point b_0 (Alg. 4).

One of the first approximate point-based algorithms, called PBVI, was proposed by *Pineau et al.* [128]. PBVI has features distinctive from its previous approximated algorithms. First, the set of reachable belief states B_r is increased gradually during the run-time. Second, the algorithm is implemented as anytime style. That is to say the quality of its solution is improved over time. PBVI can find a good solution for the Tag problem ($|S| = 870, |A| = 5, |Z| = 30$) [128] in 50 hours. One year later, *Spaan and Vlassis* [160, 161] proposed another algorithm called *Perseus*. Perseus uses a fixed set of belief points B_r. These points are sampled by taking a random-walk into

[7]The experiments were conducted on a Sun SPARC-10 computer as mentioned in [41].

Algorithm 4: approximateVI($\Gamma_0, B_r, \gamma, \epsilon$) (Approximate VI for POMDPs)

```
Input: Initial set of vectors $\Gamma_0$, set of reachable belief states $B_r$, discount
       factor $\gamma$, and positive number $\epsilon$
Output: Set of useful vectors $\Gamma'$
begin
    $\Gamma \leftarrow \Gamma_0$; $\delta \leftarrow \frac{\epsilon(1-\gamma)}{2\gamma}$; $r \leftarrow \infty$;
    while $r > \delta$ do
        $\Gamma' \leftarrow$ update($\Gamma, B_r$);
        $r \leftarrow \max_{b \in B, \alpha \in \Gamma, \alpha' \in \Gamma'} |\alpha' b - \alpha b|$;
        $\Gamma' \leftarrow \Gamma$;
    return $\Gamma'$;
end
update($\Gamma, B_r$)
begin
    $\Gamma' \leftarrow \emptyset$;
    foreach $b \in B_r$ do
        $(a^*, \alpha^*) \leftarrow backup(\Gamma, b)$;
        if $(a^*, \alpha^*) \notin \Gamma'$ then $\Gamma' \leftarrow \Gamma' \bigcup (a^*; \alpha^*)$;
    return $\Gamma'$;
end
backup($\Gamma, b$)
begin
    $\alpha'_{a,z} \leftarrow \text{argmax}_{\alpha \in \Gamma} \alpha b^a_z$;
    $\alpha'_a \leftarrow r_a + \gamma \sum_{z \in Z} \alpha'_{a,z} W^a_z$;
    return $(a^*; \alpha^*) \leftarrow \text{argmax}_{\alpha'_a} \alpha'_a b$;
end
```

the reachable belief space, starting from the initial belief state. Perseus was reported to solve the Tag problem in less than 30 minutes. In (the year) 2005, *Smith and Simmons* [157] proposed a new algorithm called *HSVI2* which was based on their *HSVI1* algorithm [156]. HSVI2 computes both lower bound and upper bound of the optimal value function using heuristic search techniques [157]. HSVI2 solved the Tag problem in only 24 seconds. They also proposed a more complex benchmark problem called RockSample [157]. HSVI2 can scale to solve a POMDP problem greater than 10^5 states (RockSample(10,10) with $|S| = 102401, |A| = 19, |Z| = 2$). The benchmark of the Tag problem is shown in Table 2.7. In Chapter 5 and Appendix A, we will report our practical tests for different dialogue management problems using Perseus and HSVI2. Recent PBVI algorithms that refine the scaling up issue and the performance are reported in *Shani et al.* [153], *Virin et al.* [170].

Method	Goal%	Reward	Time(s)	Number of α-vectors
PBVI	59	-9.18	180880	1334
HSVI1	n.a.	-6.37	10113	1657
Perseus	n.a.	-6.17	1670	280
HSVI2	n.a.	-6.36	24	415

Table 2.7: Performance comparison of the Tag problem ($|S| = 870, |A| = 5, |Z| = 30$)

To tackle larger POMDP problems, Poupart has recently implemented an algorithm called *Symbolic Perseus*. This algorithm, based on Perseus, reduces the dimensionality of α-vectors by using Algebraic Decision Diagrams [12]. Symbolic Perseus has been used to solve a real-word application where the state space is composed of 50 million states [78]. In Chapter 5 we will discuss in detail how to find a near-optimal policy for a number of dialogue problems [31, 71, 79].

Chapter 3

Dialogue management for single-application systems

This chapter is written based on Bui et al. [28]. The work was conducted (at the Artificial Intelligence Laboratory, Swiss Federal Institute of Technology in Lausanne) in collaboration with Martin Rajman. The content focuses on the work related to my own contributions that are stated in Chapter 1.

3.1. Introduction

Building a usable dialogue system is a challenging task. Many approaches to modeling dialogue have been proposed as described in Chapter 2. However, due to the complexity of the management of spoken language interfaces and their strong dependence on the interaction context, there is not yet a really generic approach for dialogue design; each application requires the development of a specific dialogue model. Dialogue prototyping, therefore, represents a significant part in the development process of interactive systems, especially for the ones with a vocal interface. In other words, there is a strong demand for an efficient rapid prototyping methodology.

This chapter presents a *Rapid Dialogue Prototyping Methodology (RDPM)* for the development of dialogue models for *single-application systems*. A single-application system is a dialogue system designed to fulfil a particular task from a predefined set of tasks that are bound to the application[1]. The general idea underlying the proposed RDPM is that the dialogue model is a frame-based model that can be quite easily and systematically derived from a relational representation of the application itself, hereafter called the *task model*. More precisely, the RDPM consists of five

[1]A precise definition of the terms *application* and *task* is presented in Section 3.2. Two tasks are belonged to the same application if they are closely related. However, this depends on the designer's opinion and requirements about the system underdevelopment. For example, "booking a flight" and "booking a hotel" might be connected to the same application (e.g., travel booking) or to two different applications (e.g., flight booking application and hotel booking application).

main consecutive steps: (1) *producing a task model* for the targeted application; (2) automatically *deriving an initial dialogue model* (and the associated dialogue-driven interface) from the produced task model; (3) using the generated interface to *carry out Wizard-of-Oz experiments* (i.e. dialogue simulations) to improve the initial dialogue model; (4) *carrying out an internal field test* to further refine the dialogue model (reformulation of system messages, improved feedback, etc.), and validate the evaluation procedure (coherence, understandability); and (5) *carrying out an external field* test to evaluate the final dialogue model.

Steps 1, 2, and 3 are the core components of the methodology. They are presented in detail in Sections 3.2, 3.3, and 3.4, respectively. Steps 4 and 5 are reported elsewhere [28, 136]. For the purpose of clarity, we again use the Restinfo system presented in Chapter 2. A case study of a more complex spoken dialogue system is presented in Section 3.5.

3.2. Producing the task model

In the RDPM framework, a task is modeled as a frame the slots of which represent the various attributes that need to be informed for the task to be performed (e.g., [107]). More precisely, a task is defined as a function the arguments of which correspond to the above-mentioned attributes and the call to which results in the fulfillment of the task. An application is seen as a set of functions the user can invoke through the dialogue-driven interface to perform various functionalities that are provided by the application. In this perspective, an application is modeled as a set of relational tables, where the rows correspond to the possible functions (also called “solutions" or the “targets") and the columns are the attributes needed to uniquely identify each of the functions, and to invoke them.

In other words, the values of the attributes in a row of the solution table (also referred to as *canonical values* or *concepts*) correspond to the values of the arguments of the function, the call of which results in the fulfillment of the corresponding application functionality. For example, the task model of the RestInfo system can be represented as a single generic function `selectRestaurant(typeOfFood, location, openingTime, openingDay, priceRange)`, the attributes of which identify the five selection features available for the restaurant search. Therefore, the task model of the RestInfo system is simply a table with five attributes: *Type of food*, *Location*, *Opening time*, *Opening day*, and *Price range*. The rows of these attributes are the various value combinations of the attributes corresponding to existing restaurants. At the computational level, the calls to the `selectRestaurant()` function are implemented in the form of SQL queries to the project database containing the required information.

Notice that the current version of the RDPM presupposes that the task model consists of a single relational table, also called the *solution table* (e.g., Table 3.1).

However, in the case of complex models consisting of several interconnected tables (for example a main table containing the acceptable value combinations of the main attributes and several additional tables relating the values present in the main table

Type of food	Location	Opening time	Opening day	Price range
Chinese	Museum	11h-22h	Mon-Sat	Cheap
Italian	Center	11h-23h	Tue-Sun	Average
French	Bourg	18h-23h	Mon-Fri	Cheap
...	...	...	...	...

Table 3.1: Excerpt of the RestInfo solution table

to additional attributes), standard database normalization procedures such as joint operations can be first applied to transform the original tables into a single large one.

3.3. Deriving the initial dialogue model

A dialogue model is defined as a set of interconnected *Generic Dialogue Nodes (GDNs)* (e.g., [18]), where each of the dialogue nodes is associated with one of the attributes (also called "slots" or "fields") in the solution table. For any given slot, the role of the associated GDN is to perform a simple interaction with the user to obtain a valid value for the associated attribute.

In the architecture that we have selected for implementing our dialogue-driven interfaces, the processing of the GDNs (i.e. the actual interaction with the user according to the specification of the GDNs) is performed by a specific module called the *local dialogue manager*. However, this is not sufficient to carry out any real dialogue: some form of global dialogue management also has to be integrated. For example, in addition to the definition of the GDNs and the specification of the local dialogue manager, a branching logic responsible for the management of the global dialogue flow needs to be specified. In the current implementation, this branching logic is hard-coded in a specific dialogue management module, called the *global dialogue manager*. We assume that the encoded local and global dialogue flow management strategies are application-independent. In most situations, they lead to an acceptable, though not always optimal behavior for the system. Consequently, in our approach, dialogue model design essentially reduces to the application-dependent, declarative specification of the GDNs where the encoded dialogue management strategies are being used without modification for all applications.

In short, a dialogue model consists of two main parts: (1) the application-dependent, declarative specification of the GDNs; and (2) the application-independent (local and global) dialogue flow management strategies encoded in the corresponding (local and global) dialogue manager. Both of these components are described in more detail in the next sections.

3.3.1. Generic Dialogue Nodes

To deal with the various attributes appearing in the solution table defining the task model, we consider three main types of GDNs:

1. Simple GDNs (also called static GDNs) associated with *static fields*, that is to say the fields of which the values do not change over time, or only change very slowly; for example the price range for a given restaurant[2];

2. List processing GDNs (also called dynamic GDNs) associated with *dynamic fields*, that is to say fields the values of which quickly change in time; for example the types of food in a selected restaurant;

3. Internal GDNs used to perform the interactions that are required by various special functions implemented in the dialogue manager (e.g. confirm a selected solution, start/reset the dialogue, etc.).

As already mentioned, the role of each GDN is to perform a simple interaction with the user to obtain a valid value for the associated attribute. In this perspective, the difference between static and dynamic GDNs is that the former expect the user to provide a value for the associated attribute directly. For example, a static GDN associated with the "Price Range" attribute might ask a question such as *"What price level are you ready to accept for a meal at the restaurant to be selected?"* and will be expecting an answer containing a value taken from a predefined list of values such as *"Cheap"*, *"Average"* or *"Expensive"*. On the other hand, a dynamic GDN will ask the user to choose from a dynamically computed list of values. For example, a dynamic GDN associated with the "Type" attribute will generate a list of meal types and ask the user to indicate the position of their selection in the list. The list processing GDNs are an important component of the targeted dialogue model as they allow us to take into account dynamic vocabularies that could not be reliably processed by simple GDNs because of the limited performance of the speech recognition module in such conditions.

To realize the interaction for which it is responsible, each GDN contains two main types of components: prompts and grammars (Figure 3.1).

Prompts

The prompts are the messages uttered by the GDN during the interaction. Several types of prompts are defined, among which are the main prompt, corresponding to the initial question asked by the GDN, and the help prompt that is uttered or visualized if a request for help is expressed by the user. The formulation of the prompts plays an important role during the dialogue. In particular, it influences the level of *mixed initiative* (i.e. the degree of flexibility that the system allows in the interaction). For instance, a main prompt such as *"What can I do for you?"* expresses the fact that the system is ready to accept a quite broad range of user requests, while a more precise prompt such as *"Do you agree to select the Chinese restaurant near the center, yes or no?"* implies a low level of mixed initiative as the user is only expected to answer with either *yes* or *no.*

[2]Note that prices might be changed quite often, but the values of the price range are usually fixed. For example, in the Restinfo system, the price range is composed of three values: cheap, average, and expensive.

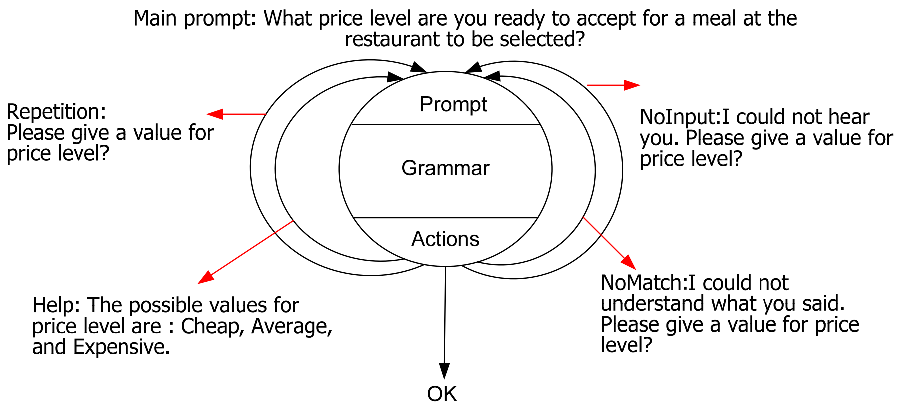

Main prompt: What price level are you ready to accept for a meal at the restaurant to be selected?
Repetition: Please give a value for price level?
Prompt
Grammar
Actions
NoInput:I could not hear you. Please give a value for price level?
Help: The possible values for price level are : Cheap, Average, and Expensive.
NoMatch:I could not understand what you said. Please give a value for price level?
OK

Figure 3.1: Example of a generic dialogue node “Price Range"

Grammars

The role of the grammars is to make a connection between the surface forms appearing in the natural language utterances made by the user and the canonical values used in the task model (i.e. the set of values defined for the attributes associated with the GDNs in the solution table for the application). As such, the grammars represent the main Natural Language Processing (NLP) elements in the system, and they might also be used in the speech recognition engine to improve the quality of the recognition.

In addition, the control of the level of mixed initiative is implemented through the notion of *active grammars.* that is to say each GDN is associated with a set of grammars that define the types of answers that are considered acceptable for the interaction that the GDN is responsible for. For example, the GDN associated with the “Time" field will be associated with the “Time" grammar (recognizing utterances such as *“at 8 o'clock"*, *“in the afternoon"*), but might also be associated with other grammars such as the “Day" grammar in order to be able to extract, in the case of an utterance such as *“tomorrow evening"* not only the time (*“evening"*) but also the date (*“tomorrow"*). Eventually, the first of the active grammars is considered as the active grammar *in focus* (i.e. the one that will be given precedence in case of ambiguity).

Finally, each GDN is always associated with some specific global grammars such as the help and repetition grammars. These grammars correspond to *Request for Help* and *Request for Repetition* situations which are mentioned in the next section.

3.3.2. Local dialogue flow management strategy

Each GDN is able to locally process five types of generic situations: (1) *OK*: the user answers the question in an acceptable way; (2) *Request for Repetition*: the user asks for repetition of the last system prompt; (3) *Request for Help*: the user does not know how to answer the question and asks for more explanation; (4) *NoInput*:

the user produces no utterance; and (5) *NoMatch*: the user answers but nothing useful can be extracted from the produced utterances. Notice that, for the cases where the user's answer is ambiguous or uncertain, the ambiguity is handled by the grammars associated with the GDNs as presented in Section 3.3.1. For example, in a travel booking application, if the user provides a value (e.g., city name) that might be belonged to either the Departure GDN or the Destination GDN. A solution for this case is as follows:

- If one of these two GDNs is in focus, then the value is only assigned to this GDN[3];
- If the two GDNs are active (but not in focus), then the value is assigned to the GDN coming first in the order defined in the solution table;
- If none of these two GDNs are active, then the value is discarded.

The presented solution to handle the ambiguous and uncertain answer is of course not optimal. Other recent approaches from the literature are given in [175, chap 4]. A promising solution is to model dialogue as a Partially Observable Markov Decision Process (POMDP). This will be presented in Chapter 5 and 6.

In the case of the *OK* situation, control is simply handed back to the global dialogue manager which applies the global dialogue management strategy for the activation of the next GDN. In the other four situations, there is a need to repair the dialogue, hence, control remains at the GDN level. In these "problematic" cases, the system operates in the following way: (a) *Request for Repetition*: the current GDN is reactivated and its main prompt is played if it is the first request for repetition, otherwise a reformulated prompt is played; (b) *Request for Help*: the GDN is reactivated and the associated help prompt is played instead of the main prompt; and (c) *NoInput/NoMatch*: the current GDN is reactivated and the NoInput/NoMatch prompt is concatenated at the beginning of the main prompt.

Notice that, in all cases, there is an upper limit to the number of consecutive times that a given GDN can be activated. If this limit is exceeded, control is handed back to the global dialogue manager with an appropriate error message.

3.3.3. Global dialogue flow management strategy

The Global Dialogue Flow Management consists of five complementary strategies:

- a branching strategy (also called *branching logic*) defining the next GDN to be activated;
- a *dialogue dead-end management strategy* to deal with dialogue situations where no solution corresponds to the request expressed by the user;
- a *confirmation strategy* to provide the system with validation possibilities for the values acquired during the interaction;

[3]There is only one focus GDN at a time.

- a *dialogue termination strategy* to define when the interaction with the user should be terminated (i.e. a solution proposed); and
- a *strategy to deal with incoherences* (i.e. situations where there are at least two incompatible values provided by the user).

As already mentioned, all these strategies are encoded in the global dialogue manager and are, therefore, application-independent. These strategies are explained in the next sections. Additionally, our dialogue manager can work either in system driven or in mixed initiative mode. We can therefore use the information from the passive modality (i.e. emotion recognition) to automatically and smoothly alternate between the two modes to enhance natural communication between the system and the user.

Branching Logic

The proposed branching logic only relies on the fact that the task model is expressed in the form of a relational table. It consists of the following four steps:

1. Acquire: some canonical values are obtained from the user through the interaction with the current GDN;
2. Filter: the obtained values are added to the set of already acquired ones and the current solution table is filtered in order to contain only the solutions that are compatible with the current set of values;
3. Propagate: for the attributes of which all the current solutions have the same canonical value, the value is propagated, that is to say considered as "implicitly" acquired for the attribute;
4. Activate: the next "open" attribute (i.e. the first attribute in the current solution table still associated with a heterogenous set of values) is identified, and the associated GDN is activated.

For example[4]:

S_1: *What do you want to do?*
U_1: *Find a Chinese restaurant near the center.*
S_2: *For Chinese as type of food and center as location, <u>which day of the week</u>?*

1. Acquire: The canonical values extracted from the U_1 utterance are: Type of food = *"Chinese"*, Location=*"Center"*;
2. Filter: If we assume that the solution table only contains three solutions compatible with the above acquired values, after filtering, the current solution table will reduce to these three solutions, as illustrated in table 3.2.

[4]In all examples, utterances identified by S_i (resp. U_i) correspond to the utterances produced by the system (resp. the user) in the i^{th} turn.

Type of food	Location	Opening time	Opening day	Price range
Chinese	Center	Evening	Sun	Cheap
Chinese	Center	Evening	Sat	Cheap
Chinese	Center	Evening	Mon	Cheap

Table 3.2: An example of a filtered solution table.

3. Propagate: the value *"Cheap"* and *"Evening"* are propagated for the "Price range" and "Opening time" attributes respectively;
4. Activate: the next "open" attribute is "Opening day" (still active with three different values); therefore, the associated GDN becomes the new active one and the question S_2 is asked.

Dialogue dead-end management

This strategy is required to deal with cases where the current solution table is empty (i.e. no solution corresponds to the canonical values acquired from the user). To cope with dead-end situations, we use the following relaxation strategy:

1. Determine how many solutions are compatible with all the values that have been explicitly acquired (i.e. not propagated) but one. If the obtained number of solutions is smaller than or equal to a predefined threshold called the *dead-end management threshold*, then provide all the relaxed solutions to the user and ask him/her to select the desired one. Otherwise, choose one of the attributes corresponding to a non-zero number of solutions when relaxed;
2. Remove the value associated with the selected attribute, re-propagate from the remaining ones, and activate a yes/no GDN to get the user's decision about the relaxation;
3. If the user agrees with the relaxation, activate the next GDN according to the standard activation rule, otherwise, go to step 2;
4. If the user rejects all relaxation possibilities, reset the dialogue.

For example:

S_1: *What do you want to do?*
U_1: *Find a Chinese restaurant near the museum.*

At this stage of the dialogue, the system has acquired the canonical values *"Chinese"* for "Type of food" and *"museum"* for "Location". If we assume that there is no Chinese restaurant near the museum, the system is in a dead-end situation. It then first relaxes *"Chinese"* and computes the number of solutions (say three) compatible with the unique constraint "Location=museum" and then relaxes *"museum"* and computes the number of solutions (say five) compatible with "Type of food=Chinese". The total number of relaxed solutions is, therefore, eight and:

- if the dead-end management threshold is greater than or equal to eight, the system will display/utter these eight solutions and ask the user to select one:

  S_2: *Your selection does not correspond to a possible solution. Please select a possible solution from the list!*

- otherwise, the system will start the relaxation confirmation process:

  S_2: *Your selection does not correspond to a possible solution. Do you agree to reconsider "museum" as location?*

Confirmation

The confirmation strategy is the procedure used during the dialogue to obtain confirmation from the user for the values that have been acquired by the system. There are two complementary approaches:

- Explicit confirmation: the confirmation is obtained by explicitly asking the user;

  For example:

  S_1: *Now you can select your restaurant from the list.*
  U_1: *I select the first one.*
  S_2: *You have selected the first solution. $\underline{\text{Is that correct?}}$*

- Implicit confirmation: the confirmation is induced from the reaction of the user to an automated confirmation information associated with the next question.

  For example:

  S_1: *Which day of the week do you want to go to the restaurant?*
  U_1: *Monday.*
  S_2: *$\underline{\text{For Monday as day}}$, what type of food are you looking for?*
  U_2: *Chinese.*

  In this example, the fact that the user did not react negatively to the indication *"For Monday as day,..."* is interpreted as an (implicit) positive confirmation. The underlying hypothesis is that the user would have reacted in a negative way (by saying something like: *"I didn't say Monday!..."*) if the proposed choice has not been correct.

Implicit confirmation usually leads to shorter dialogues and is often considered as more natural by the user. Explicit confirmation is useful in special cases, such as the invocation of irreversible actions (e.g. *"cancel my reservation in the selected restaurant"*).

Dialogue termination

The idea behind the dialogue termination strategy is that it might be more efficient, once a limited set of solutions have been reached through dialogue, to simply display/utter the solution list and let the user choose the correct one, instead of trying to continue the dialogue to refine the user request in order to reduce the solution set to a unique solution. For example, when the user wants to find a restaurant far from the city, and the number of available restaurants is sufficiently small, the system will display all the possibilities and ask the user to select a restaurant, instead of asking for additional selection criteria (e.g. the type of food). To implement such a behavior, a limit on the number of current solutions, called the *termination threshold* is defined. The termination strategy then simply corresponds to stopping the dialogue and switching to explicit solution selection as soon as the current number of solutions is smaller than or equal to the termination threshold. Once the selection is done, the task is completed and the dialogue is reset.

Incoherences

This strategy is necessary to deal with the cases where the user provides (at least two) incompatible values for one or several attribute(s). For example, if the user first indicates *"Monday"* for the Day attribute and later on provides the value *"Saturday"*, an incoherence occurs for this attribute. To deal with such situations, an incoherence prompt, such as, for example: *"I am not sure about the $\underline{\text{day}}$ you have provided: $\underline{\text{Monday}}$ or $\underline{\text{Saturday}}$. Could you please repeat your choice?"* is generated in order to force the user to make an explicit choice.

The above mentioned incoherence management strategy is only used for incompatible values that were all explicitly provided by the user (i.e. "true" incoherences). If only propagated values are involved, the new value is used to overwrite the old one. In the remaining cases (propagated against the given or vice versa), a dialogue dead-end management is triggered.

In the case of several simultaneous incoherencies, only one is processed and all other new values that lead to incoherences are removed. The rule to choose the incoherence pair to process is the following:

1. If the current GDN defines a context (i.e. is associated with a specific attribute on which the current question was focused) and if there is an incoherence associated with that attribute, then this incoherence should be processed;
2. Otherwise (e.g. in the case of GDNs corresponding to fully open questions such as: *"What do you want to do?"*), the incoherence corresponding to the attribute associated with the GDN coming first in the order defined in the solution table should be processed.

Notice that the used incoherence prompts are in fact automatically generated from prompt patterns (such as *"I am not sure about the $attributeName you have provided:*

$attributeValue1 or ...") that contain references to the attribute name and values to be used.

3.3.4. Generating the initial dialogue model and dialogue-driven interface

Each attribute in the solution table is associated with a GDN in the dialogue model, the type (static or dynamic) of which is derived from the type of the attribute. Notice that the dialogue model might also contain some GDNs that are not explicitly associated with an attribute in the solution table. For example, the initial (mixed initiative) Start GDN as mentioned in 3.3.1.

The generated GDNs are minimalist in the sense that their prompts are automatically generated and that they do not contain any interpretation grammars (or, possibly, only word spotting grammars limited to the canonical values of the associated attribute). The generated prompts are produced in a very simple way: if the attribute to be queried about is X, then the associated question is simply: *"What X?"*. Following this trivial generation procedure, we can now produce, as the final part of our illustration, an example of what could be an interaction between a user and the automatically generated dialogue model that is described in this section:

S_1: *What type of food?*
U_1: *French.*
S_2: *For French as type of food, what location?*
U_2: *Center.*
S_3: *For Center as location, what time?*
U_3: *Evening.*
S_4: *For Evening as time, what day?*
U_4: *Tomorrow.*
S_5: *Searching French restaurants near the center on Friday evening, January 31st*

In reality, if the generated dialogue model does not contain any interpretation grammars, it will, in fact, not be able to produce a dialogue, such as the one shown above, because it will be unable to interpret any of the user's answers. If the initial grammars are the automatically generated word spotting grammars, the dialogue-driven interface is operational, but, for any realistic validation, it will still require human intervention to replace the missing (or too limited) interpretation grammars. Validation is then done in the framework of Wizard of Oz (WoZ) experiments as described in the next section.

3.4. Improving the initial dialogue model using WoZ experiments

3.4.1. WoZ experiments

A WoZ experiment [70] is defined as a simulation of a human-machine interaction, during which a user is exposed to a system they believe to be fully automatic, while a hidden human operator (the wizard) is manually operating (at least) some of the system functionalities that have not yet been fully implemented (sometimes, no implementation at all has been done at the WoZ stage and the experiment then corresponds to a complete simulation) [54, 72].

Within such a setup, the main goal of a WoZ experiment is to provide “realistic" experimental data about the “true" behavior of the members of some targeted user group when faced with an automated system for a given application. To this end, the experimental data are gathered (the experiments are often recorded or filmed) and analyzed by the system designers, in order to obtain valuable insights to guide subsequent modeling and implementation decisions [55].

The underlying hypothesis is that it is easier (quicker, cheaper, etc.) to set up and modify a manually operated simulation than to specify, develop, and modify a real implementation of a system. While this hypothesis is undoubtedly very often true, it is, however, important to notice that a WoZ experiment is by no means a cheap operation. It is not an easy task to convince the user that they are really faced with a machine when the simulation is in fact operated by a human. All clues that could reveal the presence or intervention of the wizard must be thoroughly removed. Thus, actions need to be taken to physically hide the wizard during the interaction and an interaction interface, even simplified, usually needs to be developed to this end. Furthermore, the wizard has to undergo a specific training so that they can consistently behave in a manner that is as convincing as the one the user is expecting from a machine (no sophisticated inferences, no emotional reactions, no apparent tiredness, etc.). In addition, it can be quite difficult to guarantee that the behavior of the simulated system will remain uniform over time (the wizard can be in better shape one day than the other, they might not consistently remember the provided instructions to operate the simulation over a longer period of time, etc.) [107].

A WoZ experiment can significantly improve the design of an interactive system (e.g. the design of the user interface [23]). Indeed, the results of the experiments are not only used for a first evaluation of the adequacy of the initial decisions (architecture, functions, interface, etc.) of the designers for the targeted system; in addition, the experimental data gathered during the experiment can also serve, if thoroughly recorded, as initial and strongly relevant training data for the system.

The general experimental setup of WoZ experiments consisted of a wizard interacting with the user via prerecorded messages. The messages were selected by the wizard on the basis of dialogue management rules operating on the keywords extracted from the utterances of the user. The extraction of keywords was performed by the wizard through a specific interface, the WoZ interface (see section 3.4.2). All user utterances were recorded.

3.4.2. The WoZ interface

An interesting and practical solution to increase the ergonomics of the WoZ experiments is to integrate the WoZ capabilities directly into the dialogue development environment, that is to say in the dialogue manager automatically generated from the task model.

However, in order to do so, it is necessary to identify precisely the different entry points into the system that should be effectively provided to the wizard for potential intervention in system behavior. In the most general case, this might be an extremely difficult issue, as, in theory, the different dialogue models that can be considered might be of arbitrarily deep complexity.

In the case of the RestInfo system, however, the dialogue model was sufficiently simple to allow an alternative approach that did not require modification of the dialogue manager.

The dialogue model was completely re-implemented as a set of interconnected HTML forms, each representing one of the GDNs in the model and the branching logic was directly implemented in the form of hyperlinks between the HTML forms.

Each of the forms provided the wizard with necessary functionalities to operate the simulation (the option to play pre-recorded messages containing the various prompts, the option to store and visualize the various attribute values progressively acquired from the user during their interaction, the option to decide on what the wizard thinks the interpretation produced by the GDN grammar should have been, etc.).

As far as the dialogue management was concerned, it fully relied on the wizard who had to manually select the next active GDN or local processing (help, repetition, etc.) by clicking on the corresponding hyperlink.

One of the main advantages of this approach was that it made it possible to set up the WoZ experiment rapidly and at a low cost. However, the fact that the WoZ experiment was carried out in the form of a pure simulation (i.e. the WoZ environment was never connected with any of the running versions of the dialogue manager prototype) made the task of the wizard quite difficult (operating the simulation in real time appeared to be a task with quite a high cognitive load).

For this reason, we have implemented an extended WoZ interface that relies on a tighter integration of the WoZ functionalities in the dialogue management environment. In particular, the WoZ interface now integrates the same dialogue management strategy as the one implemented in the dialogue manager itself. This allows the wizard to rely on the system for the dialogue flow management and to fully concentrate on processing the user utterances (i.e. the transcription and/or the extraction of the adequate canonical values).

To guarantee an easy production of the extended WoZ interface, we have developed a WoZ Interface Generator which allows us to automatically create the WoZ Interface required for a given WoZ experiment (Fig. 3.4). The WoZ Interface Generator needs two types of input: the solution table and the configuration file containing the description of the GDNs (i.e. the dialogue model).

The result (i.e. the produced WoZ interface) consists of two main components: an application-independent library of HTML templates and Java Scripts common for all generated WoZ Interfaces, and an application-dependent component corresponding to a HTML interface, which allows the wizard to simulate the system in the WoZ experiments.

The main advantage of the WoZ Interface Generator is that it allows a very quick production of WoZ Interfaces that are simple to use and easy to modify, making it a very valuable tool for iterative dialogue model improvement.

3.5. Case study 1: the INSPIRE system

The INSPIRE Smart Home system was developed in an EU funded project[5]. The final version provides English and German language speech controls over a number of home devices (lights, TV, video recorder, electronic program guide, blinds, fan, answering machine) installed in a living room. The system is mixed-initiative. Three settings have been implemented for the output module: multiple intelligent devices, a single visible assistant, and an invisible assistant. In the first setting, the speech output is located near the location of the device, which gives a sense that the user is communicating directly with the device under control. In the second, a talking head is shown on the TV screen. It is responsible for operating the devices on behalf of the user. In the third, the speech output is similar to the second setting except the assistant is invisible like a “ghost" in the room. The prototype system has been used by several partners in the project to explore user behavior. In these experiments, the speech recognition module is usually replaced by a human wizard. A typical dialogue with the INSPIRE system is shown in Table 3.3.

In the following we discuss the production of the task model, dialogue model, and the WoZ interface.

Task model The task model is shown in the upper part of Figure 3.2. From the task model we derive the solution table which is composed of 13 fields: *Device name*, *Device location*, *Action name*, *Type*, *Day*, *Time*, *Main actors*, *Title*, *Show ID*, *Channel name*, *User name*, *Message*, and *Confirm action.* Each row of the solution table corresponds to an instance of the user’s command. For example, the first row corresponds to the command “switch on the fan". We use a special character “-" to denote the value of attributes that are not available for the corresponding command.

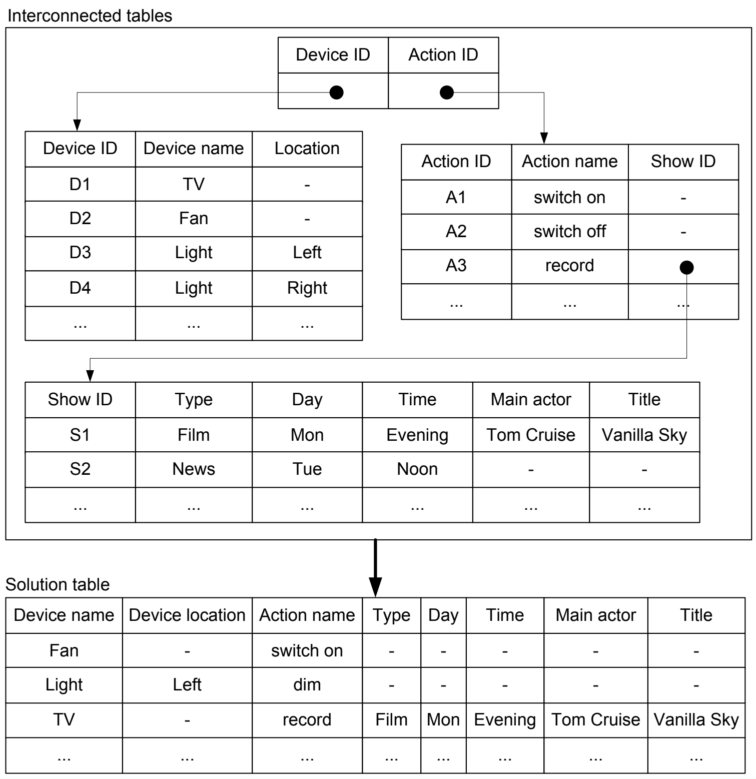
Dialogue model The dialogue model is shown in Figure 3.3. The linkage between nodes is application-dependent and can easily be specified in the XML configuration file. The bi-directional branches[6] indicate what are the other active nodes if a node is set in focus. For example, if the focus node is “User Name”, the other active nodes are

[5]IST-2001-32746

[6]Note that these branches are not the flow of the dialogue as in a finite-state dialogue model because in a frame-based dialogue model we do not need to represent the dialogue flow explicitly. In the XML configuration file of our current prototype, it is also possible to specify the branching logic branches asymmetrically.

S_1 :	Welcome to the INSPIRE smart home. What can I do for you?
U_1 :	What is on TV tonight?
S_2 :	I understood TV as device, today as day, and evening as time.
	What type of show are you looking for?
U_2 :	Film.
S_3 :	I understood films as type of show.
	Your query has returned several matches.
	Please refine your query by selecting people from the list
	[*A numbered list of people involved in films schedule tonight are shown on the TV screen*]
U_3 :	Three [*corresponding to Tom Cruise who plays in 2 scheduled films*]
S_4 :	[*A numbered list of films where Tom Cruise plays is displayed*]
	Now you can select your show from the list.
U_4 :	One.
S_5 :	What do you want me to do with this?
U_5 :	Help.
S_6 :	You can record it or set an alarm before the program starts.
U_6 :	Record it.
S_7 :	You can see a summary of your selection on the screen. Is it OK?
	[*Information about the selected show to be recorded is displayed*]
U_7 :	That's right. Thank you.
S_8 :	May I still help you?
U_8 :	Dim the light.
S_9 :	Select a location please?
U_9 :	Left.
S_{10} :	[*the light on the left is dimmed*] May I still help you?
U_{10} :	No, Thank you.
S_{11} :	I will go now. Whenever you need, I will be at your disposal.

Table 3.3: Dialogue excerpt of the interaction between the INSPIRE system and the user



Device ID	Action ID
●	●

Device ID	Device name	Location
D1	TV	-
D2	Fan	-
D3	Light	Left
D4	Light	Right
...	...	...

Action ID	Action name	Show ID
A1	switch on	-
A2	switch off	-
A3	record	●
...	...	...

Show ID	Type	Day	Time	Main actor	Title
S1	Film	Mon	Evening	Tom Cruise	Vanilla Sky
S2	News	Tue	Noon	-	-
...	...	...	...	...	...

Device name	Device location	Action name	Type	Day	Time	Main actor	Title
Fan	-	switch on	-	-	-	-	-
Light	Left	dim	-	-	-	-	-
TV	-	record	Film	Mon	Evening	Tom Cruise	Vanilla Sky
...	...	...	...	...	...	...	...

Figure 3.2: Excerpt of the INSPIRE solution table which is transformed from a set of interconnected tables. For the purpose of clarity, only eight fields of the solution table are shown.

"Device Name" and "Action Name". Some nodes have fewer branches than the other because when we merge all relational tables into one big table (i.e. solution table) we do not need to make connections between nodes, the values of which are irrelevant in the context of the application. For example, the values of the field "Location" are only used for lamps, not for the TV. Therefore, there are no connection from "Location" node to the "Main Actor" node. At the beginning, the system activates the Start GDN. It then processes the user's input based on the local and global dialogue flow management strategies presented in Section 3.3. During the interaction process, the user can end the dialogue at any time by saying "quit" command. All the current collected information will be reset. Otherwise, the system will move to the Restart GDN[7] after the task has been completed.

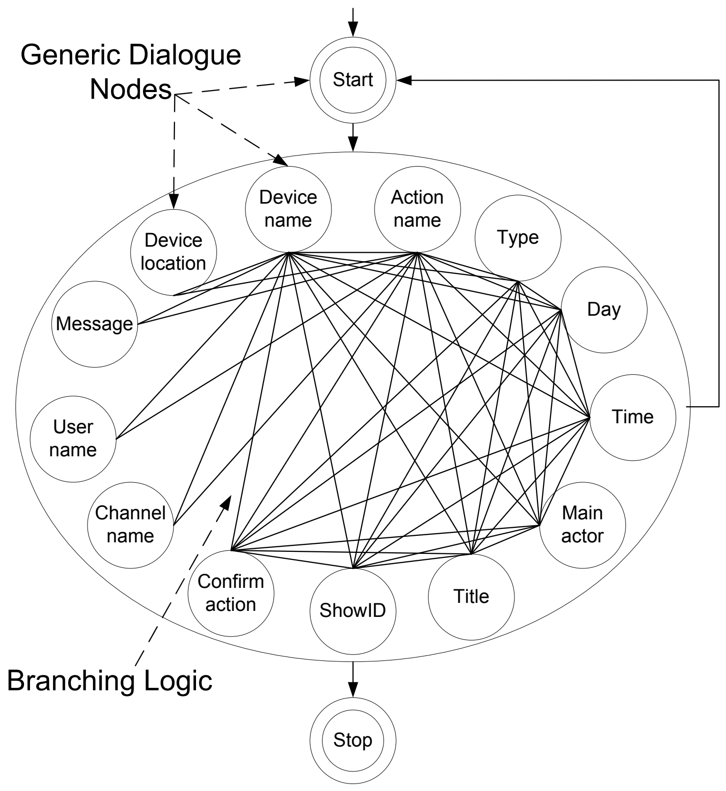
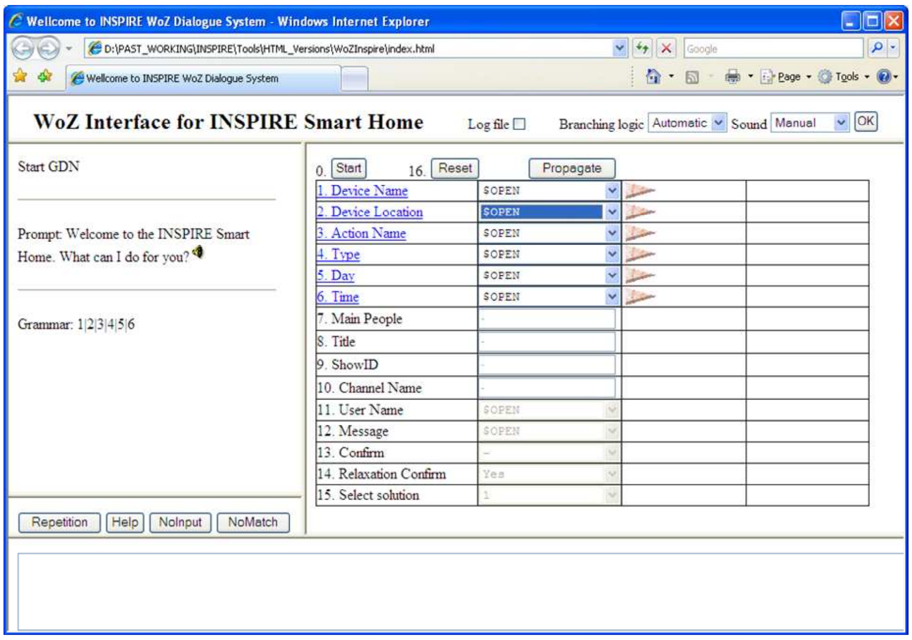


Figure 3.3: Block diagram of the dialogue model for the INSPIRE system

WoZ interface Figure 3.4 shows the first HMTL page of the WoZ interface for the INSPIRE system. These HMTL pages are automatically generated from the

[7]This functionality of this node is similar to the Start node.

solution table and the dialogue model. All the dialogue management functions are integrated. The wizard can determine either to activate the functions of the dialogue manager fully or partially depending on the experiment setup. A first WoZ experiment was conducted using this interface where the Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) modules are processed by the wizard. To enhance the development of a fully automated dialogue system, we have also implemented a second WoZ interface generator version using Java (Fig. 3.5). The application-dependent part of the WoZ interface is loaded automatically at the run-time of the WoZ interface. The Java version also allowed us to integrate different ASR and Text To Speech (TTS) modules into the WoZ interface in a flexible way. For example, it took only one working day (i.e. 8 hours) for two persons to build a Dutch language speech control of the RestInfo system by integrating the SpeechPearl speech recognition and synthesis modules for Dutch language to our RDPM software.


Wellcome to INSPIRE WoZ Dialogue System - Windows Internet Explorer
D:\PAST_WORKING\INSPIRE\Tools\HTML_Versions\WoZInspire\index.html
Google
Wellcome to INSPIRE WoZ Dialogue System
Page
Tools

WoZ Interface for INSPIRE Smart Home

Log file ☐ Branching logic Automatic Sound Manual OK

Start GDN

Prompt: Welcome to the INSPIRE Smart Home. What can I do for you?

Grammar: 1|2|3|4|5|6

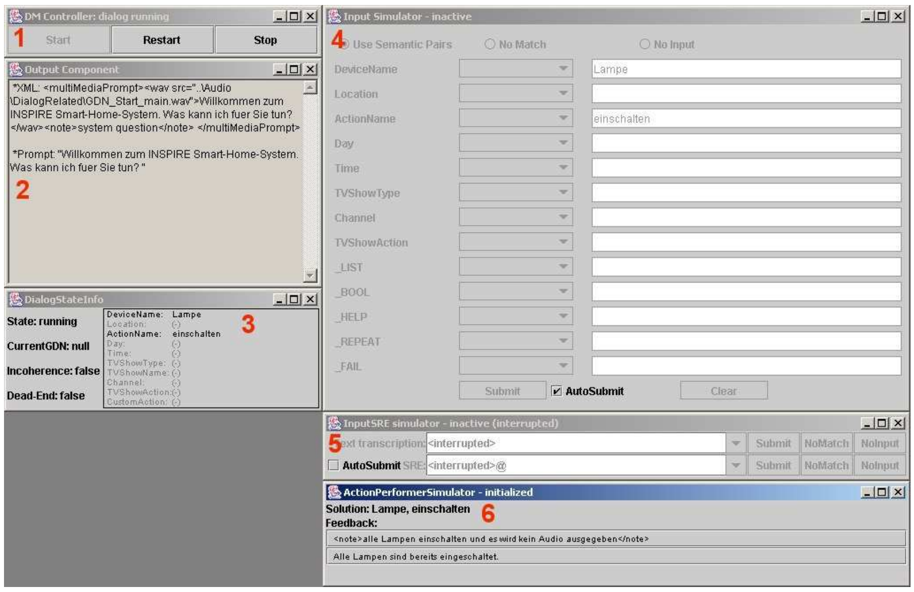
0. Start 16. Reset Propagate

1. Device Name	$OPEN		
2. Device Location	$OPEN		
3. Action Name	$OPEN		
4. Type	$OPEN		
5. Day	$OPEN		
6. Time	$OPEN		
7. Main People	-		
8. Title	-		
9. ShowID	-		
10. Channel Name	-		
11. User Name	$OPEN		
12. Message	$OPEN		
13. Confirm	–		
14. Relaxation Confirm	Yes		
15. Select solution	1		

Figure 3.4: WoZ interface for the INSPIRE system generated automatically by the WoZ interface generator

3.6. Extending the RDPM for multimodal applications

The RDPM presented in the previous sections has been developed for spoken dialogue systems. In this section, we present the extension of the methodology to build


DM Controller: dialog running
1
Start
Restart
Stop
Output Component
2
DialogStateInfo
3
Input Simulator - inactive
4
Use Semantic Pairs
No Match
No Input
Submit
AutoSubmit
Clear
InputSRE simulator - inactive (interrupted)
5
Submit
NoMatch
NoInput
ActionPerformerSimulator - initialized
6

```
*XML: <multiMediaPrompt><wav src="..\Audio
\DialogRelated\GDN_Start_main.wav">Willkommen zum
INSPIRE Smart-Home-System. Was kann ich fuer Sie tun?
</wav><note>system question</note> </multiMediaPrompt>

*Prompt: "Willkommen zum INSPIRE Smart-Home-System.
Was kann ich fuer Sie tun? "
```

State: running
CurrentGDN: null
Incoherence: false
Dead-End: false

```
DeviceName:   Lampe
Location:     (-)
ActionName:   einschalten
Day:          (-)
Time:         (-)
TVShowType:   (-)
TVShowName:   (-)
Channel:      (-)
TVShowAction: (-)
CustomAction: (-)
```

DeviceName	Lampe
Location	
ActionName	einschalten
Day	
Time	
TVShowType	
Channel	
TVShowAction	
_LIST	
_BOOL	
_HELP	
_REPEAT	
_FAIL	

☐ AutoSubmit
ext transcription: <interrupted>
SRE: <interrupted>@

Solution: Lampe, einschalten
Feedback:

```
<note>alle Lampen einschalten und es wird kein Audio ausgegeben</note>
Alle Lampen sind bereits eingeschaltet.
```

Figure 3.5: WoZ interface for the INSPIRE system (Java version)

multimodal dialogue systems.

3.6.1. Multimodal generic dialogue nodes

The generic dialogue nodes presented in Section 3.3.1 are now extended to support three active modes - text, voice, and gesture - which are used either simultaneously or independently depending on the configuration of each node. The extended generic dialogue node is called multimodal GDN or mGDN. The output of an mGDN consists of semantic pairs of the form (name, value). A local fusion mechanism therefore needs to be used inside each mGDN to combine the specific output produced by each of the input modalities and to produce the validated output semantic pairs. We also further divide the set of mGDNs into groups, where each group is considered as an object and the mGDNs in the group are attributes of the object. For example, the First name, Last name, and Function mGDNs belong to the Person group.

The prompts now contain messages and a pointing zone. The messages are either visualized in the user interface or uttered by the mGDN during the interaction or both. These messages are combined with the pointing zone (the content of which is a map, calendar or table depending on the nature of the mGDN) to allow the user to provide the desired value using keyboard, microphone, mouse click or touch screen.

3.6.2. Case study 2: The ARCHIVUS system

The extended version of the RDPM has been used to design the ARCHIVUS system [102]. It is a multimodal dialogue system that allows the user to access and retrieve the content of stored multimodal meetings, either through directed search or browsing. The user can use three input modes - voice, pointing gesture, and text - to interact with the system. The system uses three output modalities (graphics, sound, and text) to show the meeting related result to the user as well as multimedia prompts during the interaction.

The user interface itself can be divided into seven general areas, shown graphically in Figure 3.6 and explained in detail in *Lisowska et al.* [102]. In the following, we give a brief overview of these areas.

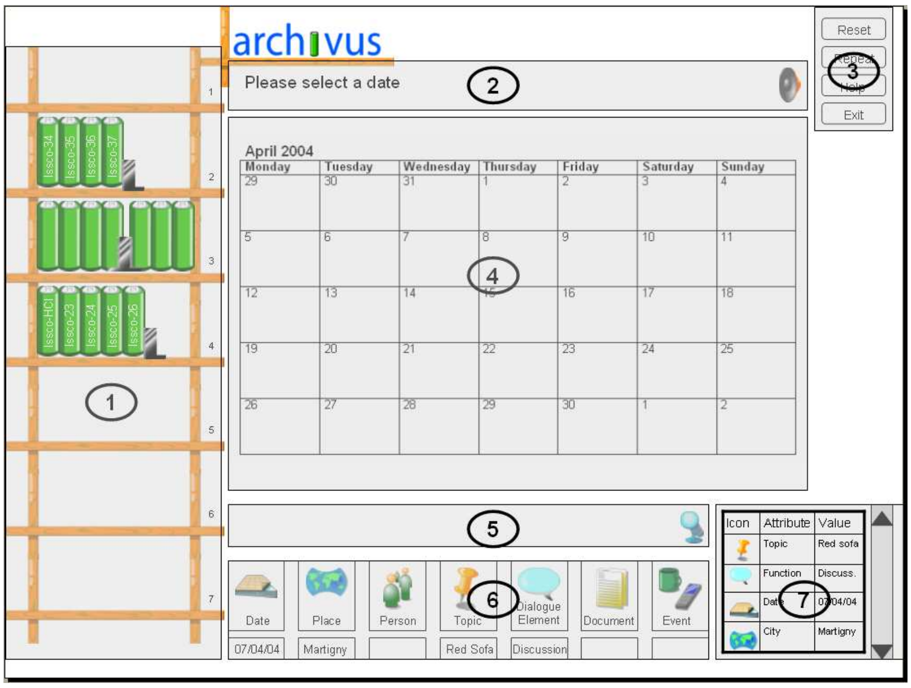

archivus
Please select a date
Reset
Repeat
Help
Exit
April 2004
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
Issco-34 Issco-35 Issco-36 Issco-37
Issco-HCI Issco-23 Issco-24 Issco-25 Issco-26
Date Place Person Topic Dialogue Element Document Event
07/04/04 Martigny Red Sofa Discussion
Icon Attribute Value
Topic Red sofa
Function Discuss.
Date 07/04/04
City Martigny
1
2
3
4
5
6
7

Figure 3.6: First design of the ARCHIVUS system [102]

1. **Bookcase**
The bookcase area serves two primary purposes. The first is to provide an overview of the entire database to the user. All of the meetings in the database will be represented as books in the bookcase. The second purpose of the bookcase is to allow the user to browse the database without having to interact with the search/dialogue engine. It should be kept in mind that browsing can be done using any of the available modalities, including voice.

2. **Prompt Bar**
   The prompt bar is used to visually display all prompts from the system. A “speaker" icon on the right-hand side indicates the availability of speech output. The user can click on this icon to deactivate and activate the speech output modality.

3. **Function buttons**
   In the upper right-hand corner of the interface there are four function buttons - three directly related to the dialogue mode (Reset, Help, and Repeat), and one system button (Exit). When the Reset button is selected, all of the constraints that have been accumulated during a dialogue are erased and the system is reset to accept a new query. The Repeat button enables the user to hear the last prompt again, in cases where they misheard or misunderstood it. The Help button provides access to online help which explains the functionalities available in the interface, while the Exit button quits the ARCHIVUS system altogether.

4. **Interactive display area**
   This area serves three purposes depending on the point of the interaction at which the user finds themselves; a visual display of the interactive dialogue mode, a space in which to view a meeting book, and a space in which to view multimedia elements of the database such as video and accompanying documents.

5. **User input box**
   The user input box is where the user can either type an information request, or, in the case of speech input, the results of the speech recognition software appear. This area effectively allows the user to initiate a natural language driven directed search, where the user already has some idea of the information that they are seeking and poses a specific request for information to the system. Additionally, the microphone icon on the right side of the box indicates the availability of voice as an input modality.

6. **Criteria refinement buttons**
   In order to help refine a request during the dialogue mode the interface is equipped with criteria refinement buttons. These buttons, selectable using any of the three input modalities, help the user refine a particular request.

7. **History area**
   A well-known HCI design principle is that the user should always be able to follow and backtrack along the path which they used to reach a particular point in their interaction. To this end we provide the history area, which shows the user, in iconified and textual form, the constraints that they have imposed in order to reach their goal. The content of the history will be represented in a scrollable pane in reverse chronological order.

3.7. Conclusions

This chapter describes the RDPM for a quick production of frame-based dialogue models for single-application systems. The practical result shows that an initial dialogue model can be developed in several hours for simple applications such as the RestInfo system. Our dialogue manager is able to handle a wide range of frame-based dialogue management functions such as branching logic, dialogue dead-end management strategy, confirmation strategy, dialogue termination strategy, incoherencies, strategy defining level of initiative.

We also implemented a WoZ interface generator that allows an automatic creation of WoZ interfaces. These interfaces are integrated with the dialogue management library, which facilitate the tasks of the wizard and the operator in a WoZ experiment.

The methodology has been used in several projects. In the Inspire project, the methodology was used to implement a more complex (than the RestInfo system) spoken dialogue system that allows the user to operate various devices in the living room. The strategies for dialogue management have been extended and validated. Several modifications were made in the core of the dialogue management (e.g. a cleaner dialogue dead-end management, a more sophisticated processing of the word spotting grammars, etc.). In addition, functions related with user modeling and system customization have been integrated. In particular: (1) *Reset Patterns* that allow the system to adapt to the behavior of a specific user or population of users by anticipating their next decisions [132], and (2) *Custom Actions* that allow the user to dynamically associate sequences of solutions with a single new solution [29]. The main goal of these extensions is to reduce the time to perform a task with the interface. The hypothesis is that these functions will indeed increase the quality of the interaction as perceived by the user. In the IM2 project, we extended the methodology for the design of the first version of the ARCHIVUS multimodal dialogue system [102]. This work provided an important background for a further implementation of the ARCHIVUS system [4, 43, 110].

3.8. Historical remarks

The RDPM was first proposed by *Rajman et al.* [134, 135] for implementing finite-state spoken dialogue models for simple applications such as the RestInfo system. Based on this framework, we have extended the methodology which allows us to build frame-based dialogue models for more complex dialogue applications [27, 28] in a short time. We also proposed and extended the RDPM for the design of multimodal dialogue management models. A first design of the ARCHIVUS system is described in [102]. The framework provided an essential infrastrutre for implementing and testing the dialogue management module of the ICIS-CHIM demonstrator (see Chap. 6). A similar work that focuses on extending the RDPM for multimodal WoZ experiments is reported in *Melichar* [109]. The further development of the ARCHIVUS system is reported in [4, 43, 109, 110].

Chapter 4

Dialogue management for multi-application systems

This chapter is written based on Bui et al. [30]. We propose a novel approach to developing multimodal interfaces for multi-application dialogue systems. The targeted interfaces allow transparent switching between a large number of applications within one system.

4.1. Introduction

The RDPM presented in Chapter 3 allows us to develop single-application dialogue systems in a short time. Each system is tailored to a specific application domain such as flight booking, train time table information, and hotel reservation. Consider the following example in the crisis management domain:

S_1	Welcome to the crisis management support center. What can I do for you?
U_1	I want a route description to the Benelux tunnel.
S_2	I understood Benelux tunnel as destination. Where are you now?
U_2	I am in the police station 1.
S_3	The route description from the police station 1 to the Benelux tunnel is shown on the map. Is it OK?
U_3	Yes. Thank you.
S_4	What else can I do for you?
U_4	Umm, I want to have some information about a patient.
	[*single-application dialogue system*]
S_{5a}	Sorry, the system only supports the route navigation tasks.
	[*multi-application dialogue system*]
S_{5b}	Seeking for the patient information. What is the patient's name?

This example shows a clear advantage of the multi-application dialogue system over the single-application counterpart. Formally, a multi-application dialogue system[1] is defined as a dialogue system allowing the user to navigate between a set of applications where each application is associated with a set of related tasks. Tasks in each application considered range from simple tasks such as operating a home device or booking a flight to more complex tasks such as controlling a smart-room or managing (road) traffic. Notice that the boundary between single-application systems and multi-application systems is not necessary disjoint. If we consider each application is composed of only one task and all tasks are related, a single application with multiple related tasks is then considered as a multi-application. In any case, the methodology that is going to be presented provides a potential solution for a transparent switching between these applications (tasks).

In this chapter, we present a generic dialogue modeling methodology for the efficient production of interfaces for multi-application dialogue systems. The targeted interface allows transparent switching between a large number of applications within one system. The approach, based on the RDPM and the Vector Space Model (VSM) techniques, is composed of three main steps: (1) producing finalized dialogue models for applications using the RDPM, (2) designing an application interaction hierarchy based on VSM techniques, and (3) navigating between the applications based on the user's application of interest.

These steps are described in Sections 4.2, 4.3, and 4.4 respectively. A scenario example for producing a dialogue system accessing ten applications in the ICIS domain is presented in Section 4.5. Possible extensions of the methodology and a recently related work are discussed in Section 4.6. Section 4.7 summarizes the main points of the chapter and possible further extensions of the methodology respectively.

4.2. Producing finalized dialogue models for applications using the RDPM

The finalized dialogue model for each application can be quickly produced using the RDPM presented in Chapter 3. In the following we discuss in detail the general architecture of the multimodal dialogue system corresponding to each single application produced by the RDPM (Fig. 4.1).

Three input modalities: voice, text and pointing can be used independently or simultaneously depending on the configuration of the current active mGDN [102]. These inputs are pre-processed by the Natural Language Understanding (NLU) modules and the Pointer Understanding (PU) module. The outputs from NLU and PU modules are semantic triples (attribute, value, time-stamp). The fusion manager integrates the semantic triples received from the NLU and PU modules and sends a set of integrated semantic triples to the dialogue manager. In the current implemented version, the fusion manager simply collects the semantic triples based on their time-stamp relation and forwards them to the dialogue manager.

The dialogue manager encodes the local dialogue flow management strategy and global dialogue management strategy. Therefore, the input to the dialogue man-

[1]A similar name that is usually used by the dialogue research community is multi-domain dialogue system.

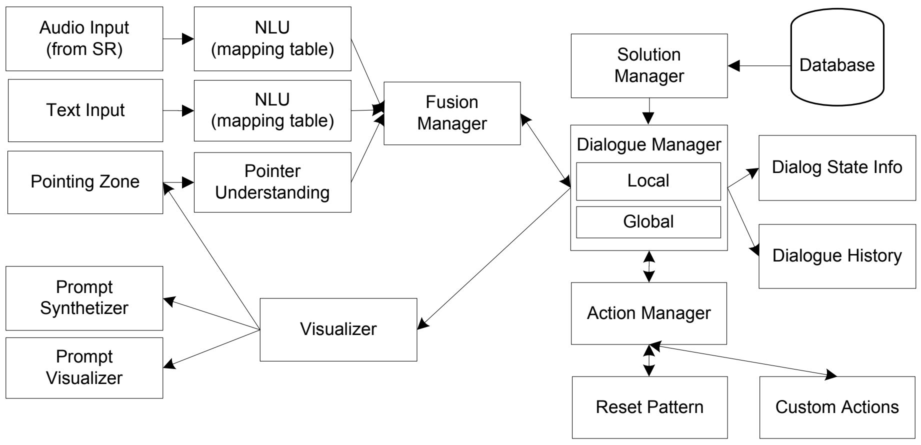


Figure 4.1: Architecture of dialogue systems produced by RDPM

ager is first processed by the local dialogue management strategy as described in Section 3.3.2.

In the case of the *OK* situation, control is handed back to the global dialogue manager which applies the global dialogue management strategy for the activation of the next mGDN. The dialogue state information (e.g. the current dialogue state, the active mGDN, etc.) and the recognized semantic triples are updated to the dialogue state info module and the dialogue history module respectively. When the dialogue manager gathers enough constraints [2], it sends the request to the action manager, the application connected with this module performs the task and sends the feedback to the action manager, the action manager then forwards this feedback to the dialogue manager. In addition, functions related with user modeling and system customization have been integrated such as *Reset Patterns* and *Custom Actions*. Reset Patterns allow the system to adapt to the behavior of a specific user or population of users by anticipating their next decisions. The idea is to develop an intelligent reset algorithm that estimates the most probable values for some mGDN slots in a new dialogue session according to the previous interactions with the user. Custom Actions allows the user to dynamically associate sequences of solutions with a single new solution. The main goal of these two functions is to reduce the time to perform a task with the interface. The hypothesis is that these functions will indeed increase the quality of the interaction as perceived by the user. These two functions are described in detail in *Bui et al.* [29].

The outputs from the dialogue manager to the visualizer are multimedia prompts containing messages and a pointing zone update content. The messages are visualized in the user interface (Prompt Visualizer) and/or uttered by the mGDN during the interaction (Prompt Synthesizer). The messages are combined with the pointing

[2]this happens when the number of solutions (extracted from the solution manger) satisfying the current constraints is smaller than or equal to a pre-defined solution threshold.

zone update content (the content is a map, a calendar or a table depending on the nature of the mGDN) to allow the user to provide the desired values using keyboard, microphone or mouse click/touchscreen.

4.3. Designing an application interaction hierarchy

Applying the RDPM, it is possible to produce n finalized dialogue models M_1, M_2, ..., M_n from n applications $A_1, A_2, ..., A_n$[3]. The question is how to integrate these applications in one unique system (i.e. multi-application dialogue system).

Vrugt and Portele [172] introduced a dialogue system accessing multiple applications with a dynamic setup that can be changed at run-time. Their goal is achieved by implementing an application-independent knowledge processing module inside the dialogue system based on modular ontological descriptions. They also define a clear interface between a dialogue system and applications by realizing a generic dialogue functionality on top of the application independent knowledge processing. This approach assumes that the user knows exactly which application he is going to interact with (similar work that uses this assumption is reported in *Seneff et al.* [152]) and therefore it is not scalable to the development of dialogue systems with a large number of applications.

Chu-Carroll and Carpenter [46] developed a call-routing dialogue system using the VSM techniques. The system allows routing the user's telephone call to the right department. Two main modules in the system are the routing module and the disambiguation module. When the routing module returns more than one candidate application, the disambiguation module is invoked. The disambiguation module determines the number of terms relevant to the user's request (say n) and uses a YN-question ($n = 1$)[4] or a WH-question ($n > 1$) to identify the desired application (i.e. the department) or transfers the call to the operator ($n = 0$). The authors do not view each application as a finalized dialogue model, therefore no further interaction happens when an application is identified.

We organize applications in an hierarchy since it allows flexible dealing with a large number of applications [52]. The hierarchy can be created manually or automatically. When the number of applications is large (hundreds, thousands, or more[5]), it is difficult to create the hierarchy manually, therefore an automatic process is suitable for this case. In our approach, the hierarchy is produced automatically using VSM techniques and an hierarchical clustering algorithm.

4.3.1. Application interaction hierarchy

An application interaction hierarchy is an m levels hierarchy of n finalized dialogue models consisting of three types of nodes:

[3]Each application can have its own set of input modalities as described in Section 3.6

[4]This case corresponds to two candidate applications. Only one term with disambiguating power is sufficient to distinguish the right application (see [46, pg. 377] for a further detail).

[5]We assume that each application is described by an associated textual document and the main goal is to find out the user's application of interest

1. Root (level m): unique node on the top of the hierarchy.
2. Internal nodes (levels from $m - 1$ to 2): each internal node consists of at least two child nodes, a child node can be an internal node or a leaf. The hierarchy accepts lattice nodes (i.e. internal nodes, each of them has more than one father node).
3. Leaves(level 1): correspond to n applications.

An application interaction hierarchy ($n = 10$) is represented in Fig. 4.4. Notice that the application interaction hierarchy is not always fully balanced such that all leaves are the same distance from the root. The level of an internal node is determined based on distance from the longest leaf that is originated from this node.

4.3.2. Vector space model for the finalized dialogue models

We assume that each finalized dialogue model of which the production is described in Chapter 3 is characterized by a textual description of the associated application. The textual description can be extracted from the mapping tables (cf. Fig. 4.1). We represent these descriptions by k-dimension vectors $d_1, d_2, ..., d_n$ using the VSM techniques.

The following paragraph presents the process of producing vectors and computing the similarity between the textual descriptions of the applications using the standard VSM technique (in the implementation phase, a suitable VSM and the number of index terms are selected based on the content of textual descriptions. For example, in the case where the textual description is a set of sentences, a semantic VSM taking into account the dependence between terms such as [133] is appropriate):

1. **Produce index terms from the textual descriptions**
   We analyze the textual descriptions using Natural Language Processing (NLP) techniques (syntactic analysis, morphological & stop words filtering, term extraction) to produce k index terms: $t_1, t_2, ..., t_k$.

2. **Construct occurrence matrix F**
   A description is represented by a lexical profile:

   $$d_i = (w_{i1}, w_{i2}, ..., w_{ik}), \tag{4.1}$$

   where w_{ij} is the weight (or importance) of the j^{th} indexing term t_j in the textual description d_i. The weight w_{ij} is often simply the number of occurrences of t_j in d_i or the inverted occurrence frequency.

   The $n \times k$ occurrence matrix F is constructed as follows:

   $$F = \begin{pmatrix} d_1 \\ d_2 \\ ... \\ d_n \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & ... & w_{1k} \\ w_{21} & w_{22} & ... & w_{2k} \\ ... & ... & ... & ... \\ w_{n1} & w_{n2} & ... & w_{nk} \end{pmatrix}$$

3. **Compute the score (or measure the similarity)**
   The most common similarity measure for the standard VSM is the cosine of the angle between the two vectors:

$$\begin{aligned} sim(d_i, d_j) &= \cos(\overrightarrow{d_i}, \overrightarrow{d_j}) \\ &= \frac{\sum_{p=1}^{k}(w_{i_p} \times w_{j_p})}{\sqrt{\sum_{p=1}^{k} w_{i_p}^2 \times \sum_{p=1}^{k} w_{j_p}^2}}. \end{aligned} \tag{4.2}$$

   We use this measure to determine the similarity between two applications, i.e. the score between A_i and $A_j : s(A_i, A_j) = sim(d_i, d_j)$.

4.3.3. Hierarchical clustering algorithm

From the vectors $d_1, d_2, ..., d_n$ and their similarity computed by Equation 4.2, we use the hierarchical clustering algorithm [85] to produce the application interaction hierarchy:

1. Consider each d_i is a single cluster, we have n clusters. The distance between a pair of clusters i and j (in this step) is:

$$D(i, j) = 1 - sim(d_i, d_j). \tag{4.3}$$

2. Find the most similar pair of clusters (i.e. $\min(D(i, j))$ and merge them into a single cluster, so that we have one cluster less.

3. Compute distances between the new cluster and each of the old clusters.

4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size n.

Step 3 can be done in several ways such as single-linkage, complete-linkage, or average-linkage clustering [84]. Applying the single-linkage, the formula to calculate the distance between two clusters C_1, C_2:

$$D(C_1, C_2) = \min_{i \in C_1, j \in C_2} [D(i, j)] \tag{4.4}$$

The output of the presented clustering algorithm is a binary tree (Fig. 4.3), this tree is transformed to an application interaction hierarchy based on the degree of similarity between the applications (Fig. 4.4). For example, if a node N_1 has two child nodes (N_2, N_3) and N_2 has two child nodes N_4, N_5 and $[D(N_2, N_3) - D(N_4, N_5)] \leq \alpha$, α is a predefined threshold, then N_2 is removed; N_4 and N_5 become child nodes of N_1.

4.4. Navigating between applications based on the user's application of interest

The system aims to find out the target application with a minimal number of dialogue turns. Based on the application interaction hierarchy produced in Section 4.3, the preliminary experimented work presented in *Forler* [68], and the textual content provided by the user, the user-system interaction process is described in detail in the following algorithm:

1. **Start**
   The system starts with a generic prompt: "What can I do for you?" (similar to the internal GDN "Start" described in Chapter 3).

2. **Active node determination**
   When receiving a user's request, the system first represents the request in the form of a vector $q = (q_1, q_2, ..., q_k)$ using the set of k index terms described in Section 4.3.2, and then determines the active node on the hierarchy by the following steps:

   (a) Score computation
   Compute the similarity between q and $d_1, d_2, ..., d_n$, we obtain a set of scores $\{s_1, s_2, ..., s_n\}$, where $s_i = sim(d_i, q)$.
   For example, in Fig. 4.2 we have $s_1 = 0.85, s_2 = 0.9, ..., s_{10} = 0.15$.

   (b) Upward propagation
   Select the best scores at each level and propagate them upward until the root is reached.
   For example, in Fig. 4.2 we have $s_{1-3} = max(s_1, s_2, s_3) = 0.9$.

   (c) Downward traversal to determine the active node
   Start from the root, compute the difference between two highest score child nodes, if this difference is below a certain threshold (we call this threshold the internal node stop threshold $t_s : (0 < t_s \leq 1)$), then stop. If not, go down to the highest score child node and continue to determine the active node.
   For example, in Fig. 4.2, starting from M_{1-10}, we calculate the difference between M_{1-5} and M_{6-10}: $dif(M_{1-5}, M_{6-10}) = 0.5$, it is greater than $t_s = 0.15$, then we go down to M_{1-5}, we still have $dif(M_{1-3}, M_{4-5}) = 0.2$ is greater than t_s then we go down to M_{1-3}, we have $dif(M_1, M_2) = 0.05 < t_s$ then M_{1-3} is the active node.

3. **Response generation**
   The active node identified in the previous steps can be a root, an internal node or a leaf. Two types of response depending on the position of the active node are:

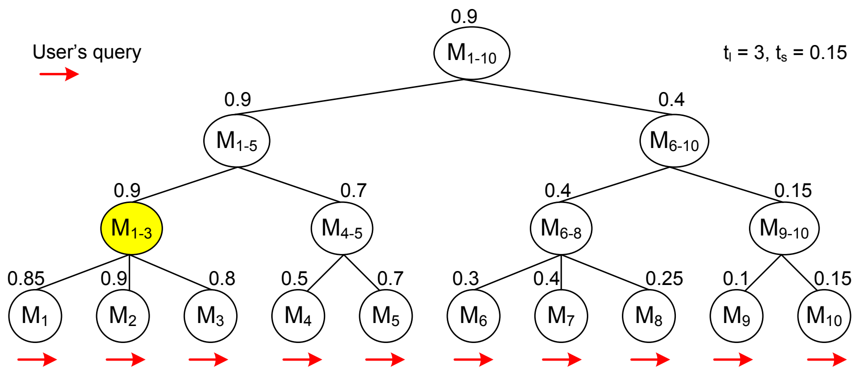


Figure 4.2: Determining the active node based on the user's query

(a) The active node is the root or an internal node
In this case, the functionality of the active node is similar to the list processing GDN described in Section 3.3.1. The system shows a list of application candidates belonging to the active node and their score is not below the highest score leaf outside the active node (e.g. in fig. 4.2, M_5 is the highest score leaf outside the active node M_{1-3}). To avoid showing a bulky list to the user (particularly in case of vocal dialogue), the maximum number of application candidates is limited by a threshold called the list processing threshold t_l, with t_l is a positive integer. The user can determine to go `next` (i.e. show the t_l following application candidates), `previous` (i.e. show the t_l previous application candidates), `stop` (i.e. restart the dialogue), `up` (i.e. move to the upper level on the hierarchy), `down` (i.e. move to the highest score child node), or `select` the desired application.
If the user does not change the active node (i.e he does not use the command up or down) and after browsing all the applications (belonging to the active node) he could not find his desired applications, the system temporarily assigns the scores of the browsed leaves to zero and goes back to step 2b.

(b) The active node is a leaf
The application takes control and interacts with the user as an application-specific dialogue system. If the user's request is out of the application domain, go back to step 2.

An example of the algorithm with $n = 10$ and $t_l = 3$ is presented in Fig. 4.5 and explained in detail in section 4.5.

4.5. Scenario example

This section illustrates, on the global level, the process of developing a dialogue system accessing 10 applications in the ICIS domain using three steps presented in the previous sections. The applications are: car route navigation (A_1), air route navigation (A_2), traffic lanes (A_3), map and fire management (A_4), tunnel sensors management (A_5), weather forecast (A_6), virtual control room (A_7), road surface temperature monitoring (A_8), patient information search (A_9), and medical worker verification (A_{10}).

Step 1 Applying the RDPM, we produce the finalized dialogue models: $M_1, M_2, ..., M_{10}$.

Step 2 From the finalized dialogue models, we create the application interaction hierarchy (cf. Fig. 4.4). Finalized dialogue models for the root and internal nodes are the list processing mGDNs produced by the RDPM. The role of each node is to select a subset of the applications belonging to it, for example the role of M_{1-3} is to select a subset of $\{A_1, A_2, A_3\}$.

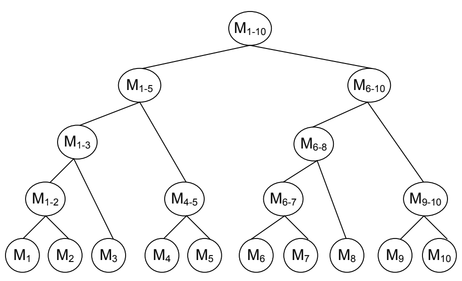


Figure 4.3: Binary tree

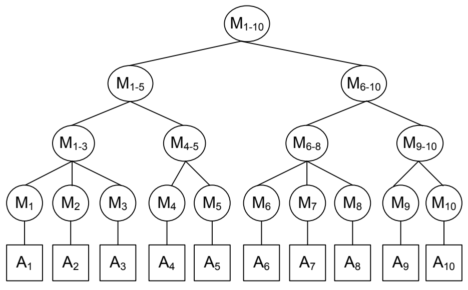
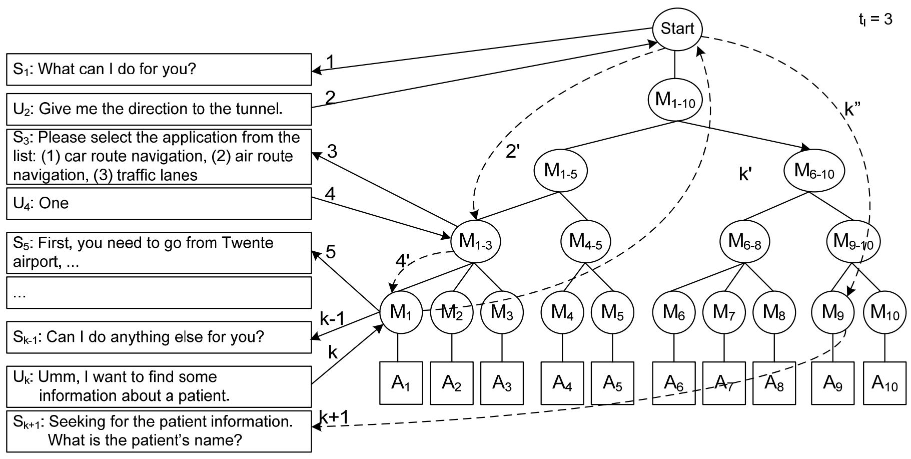


Figure 4.4: Application interaction hierarchy

Step 3 An example of the system-user interaction is presented in Fig. 4.5. The "Start" mGDN sends the system's prompt S_1 to the user. According to the content of the user's prompt U_2, the active node M_{1-3} is determined. M_{1-3} asks the user to select an application from the list $\{A_1, A_2, A_3\}$ (all three applications are shown because $t_l = 3$). Based on the user's answer in U_4, M_1 is activated. In steps from 5 to $k-1$, M_1 interacts with the user as an application-specific dialogue system. In step k, the user's request U_k is out of M_1's application domain, M_1 then forwards U_k to the system. The system analyzes U_k and activates M_9. M_9 continues the interaction with the user and processes the "out of the application domain" case in a similar manner M_1 has done.


S_1: What can I do for you?

U_2: Give me the direction to the tunnel.

S_3: Please select the application from the list: (1) car route navigation, (2) air route navigation, (3) traffic lanes

U_4: One

S_5: First, you need to go from Twente airport, ...

...

S_{k-1}: Can I do anything else for you?

U_k: Umm, I want to find some information about a patient.

S_{k+1}: Seeking for the patient information. What is the patient's name?

$t_l = 3$
Start
M_{1-10}
M_{1-5}
M_{6-10}
M_{1-3}
M_{4-5}
M_{6-8}
M_{9-10}
M_1 M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_9 M_{10}
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10}
1
2
3
4
5
k-1
k
k+1
2'
4'
k'
k"

Figure 4.5: Navigating between the applications

4.6. Possible extensions

This section discusses two main possible extensions of the generic dialogue modeling methodology: *crossing-application* and *task selection.*

4.6.1. Crossing-application

The application interaction hierarchy created in section 4.3 can be used to manage several concurrent applications (i.e. crossing-application). This extension is significant when the user wants to simultaneously execute several applications in order to achieve his goal in an optimal way. For example, in the scenario presented in section 4.5, the user's goal is to find out an optimal route for sending a rescue team to the disaster site. Suppose that the system contains two applications, the car root navigation application and the traffic lanes application. Obviously, if the user can interact with both these applications simultaneously, his goal can be more quickly satisfied than if he interacts with each application sequentially.

4.6.2. Task selection

In the definition of the application interaction hierarchy in the section 4.3.1, we mentioned that each leaf corresponds to an application. In task-oriented dialogues, each application usually consists of several tasks. We can extend the hierarchy for identifying a task or a set of tasks in an application. To achieve this goal, the hierarchy will be constructed from the set of tasks in the same way we have done for the set of applications.

Recently, Song [159] has proposed a solution for solving the task sharing issue. First, the application descriptions are merged into a unified application description. The dialogue model is then constructed based on this unified application description. She also developed software to allow the developers to combine the different application descriptions semi-automatically.

4.7. Conclusions

We have presented a framework for the development of interfaces for multi-application dialogue systems. Three important steps in the framework are described and illustrated by a scenario example.

Currently, the RDPM software toolkit is available for the development of finalized dialogue models for single applications. It has been used in two research projects: INSPIRE and IM2.MDM to validate the principle idea of the methodology, and is being extended for the development of a large number of applications in the ICIS domain. The practical result shows that from a simple application, we can develop an initial dialogue model in several hours. The dialogue manager, the most important part of dialogue prototyping, covers most of the application independent dialogue functionalities (i.e. branching logic, dialogue dead-end management strategy, confirmation strategy, dialogue termination strategy, incoherencies, strategy defining level of initiative, etc.) Therefore, we can re-use the dialogue manager and the other modules described in section 4.2 for the development of multi-application dialogue systems.

Some initial work toward developing the application interaction hierarchy and navigating between the applications (sections 4.3 and 4.4) has been analyzed and implemented (e.g. NLP Pre-Processing Tool, VSM). The multi-application dialogue system for ICIS domain presented in section 4.5 is currently under development.

Chapter 5

Affective dialogue management using factored POMDPs

Part of this chapter is presented Bui et al. [31, 33]. The chapter provides both theoretical and practical insights into applying the POMDP technique for modeling affective dialogue.

5.1. Introduction

As mentioned in Chapter 1, a challenging issue in the design of an Affective Dialogue System (ADS) is to infer the *user*'s affective state and adapt the system's behavior accordingly. This chapter addresses this issue by introducing a dialogue management system which is able to act appropriately by taking into account some aspects of the user's affective state. The computational model used to implement this system is called the *affective dialogue model*. Concretely, our system processes two main inputs, namely the observation of the user's action (e.g., dialogue act) and the observation of the user's affective state. It then selects the most appropriate action based on these inputs and the context. In human-computer dialogue, building this sort of system is difficult because the recognition results of the user's action and affective state are ambiguous and uncertain. Furthermore, the user's affective state cannot be directly observed, and usually changes over time. Therefore, an affective dialogue model should take into account basic dialogue principles, such as turn-taking and grounding, as well as dynamic aspects of the user's affect.

An intuitive solution is to extend the Rapid Dialogue Prototyping Methodology (RDPM) presented in Chapter 3 by integrating an Affect Recognition (AR) module and define a set of rules for the system to act given the observation of the user's affective state (e.g., using the OCC rules). However, it is not trivial to handle uncertainty using a pure rule-based approach such as the RDPM. A promising approach to handle uncertainty is to model the dialogue as a Partially Observable Markov Decision Process (POMDP) [142, 177]. We follow this approach to design our affective dialogue model.

In this chapter, we first introduce basic components of an ADS. Second, we give an overview of POMDP techniques and their applications to the dialogue management problem. Third, we describe our factored POMDP approach to affective dialogue modeling. Finally, we address various technical issues when using state-of-the-art approximate POMDP solvers to compute a near-optimal policy for a single slot route navigation application.

5.2. Components of an affective dialogue system

An ADS is a Multimodal Dialogue System (MDS) where the user's affective state might be recognized from speech, facial expression, physiological sensors, or combined multimodal input channels. The system expresses emotions through multimodal output channels such as a talking head or a virtual human. Figure 5.1 shows an architecture of a speech-based ADS. The speech input is first processed by the Automatic Speech Recognition (ASR) module and the semantic meaning is then derived by the Natural Language Understanding (NLU) module. Similarly, the user's affect is interpreted by the AR module and the result is sent to the Dialogue Manager (DM). The DM processes both inputs from the NLU module and the AR module and produces the system action and the system's affective state which are processed by the Natural Language Generation (NLG) module and the Text To Speech (TTS) module before sending to the user.

The detailed interaction between the system and the user is described by the following cycle[1]. The user has a goal g_u in mind before starting a dialogue session with the system. The user's goal g_u might change during the system-user interaction process. At the beginning, the DM sends an action a_s (e.g., informing that the system is ready or greeting the user) and optionally an affective state e_s (whether it is appropriate to show the system's affective state depends on each particular application). Action a_s usually represents the system's intent and is formulated as a sequence of dialogue acts and their associated semantic content. The NLG module processes the tuple $< a_s, e_s >$ and realizes a sequence of utterances w_s. The tuple $< w_s, e_s >$ is then processed by the TTS module and the outcome is an audio signal x_s. The signal x_s might be corrupted by the environment noise n_s and the user perceives an audio signal $\tilde{x}_s$. Based on the perceived signal $\tilde{x}_s$, the user infers the system's intent $\tilde{a}_s$ and the affective state $\tilde{e}_s$. Subsequently, the user maintains a belief b_u about the current system action and affective state. The tuple $< \tilde{a}_s, \tilde{e}_s >$ might be different from its counterpart $< a_s, e_s >$ due to a misunderstanding by the user or the corrupted noise n_s from the environment or both. Based on the user's goal g_u and the user's belief b_u, the user forms a communicative action (i.e., intent) a_u.

[1]This description is based on a general statistical Dialogue System (DS) framework presented in *Young* [182] and our prototype DSs presented in Chapters 3 and 4.

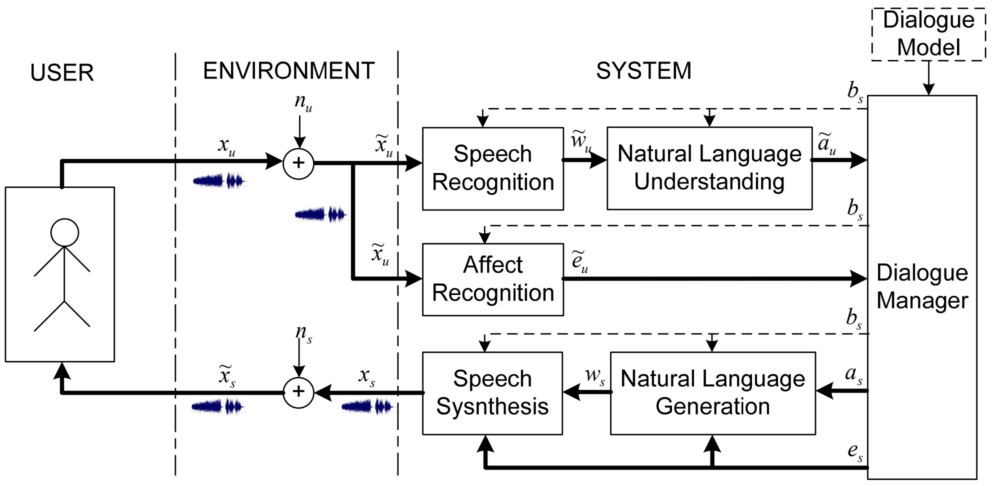


Figure 5.1: Components of an affective speech-based dialogue system. Bold arrows show the main flow of the interaction process. Dashed arrows show the links from the system's belief state to its individual modules.

Action a_u might also be influenced by the user's affective state e_u. The user then formulates a sequence of words w_u and articulates a speech signal x_u. See *Levelt* [97] for further details of the transition from intention to articulation performed by the user. The acoustic signal x_u is processed by both the ASR module and the AR module (the actual input of these modules is, $\tilde{x}_u$, a corrupted signal of x_u caused by the environment noise n_u). The output of the ASR module is a string of words $\tilde{w}_s$. This is then processed by the NLU module and the result is the observation of the user's action $\tilde{a}_u$. The output from the AR module ($\tilde{e}_u$) and from the NLU module ($\tilde{a}_u$) are sent to the DM. The DM selects a next system's action based on these inputs and the current system's belief b_s. The process is then repeated.

5.3. Review of the POMDP-based dialogue management

Section 2.5 explained the basic activity of a POMDP-based dialogue management system. *Young et al.* [183] have argued that nearly all existing dialogue management systems, especially those based upon the information state approach [165], can be considered as direct implementations of the POMDP-based model with a *deterministic* (i.e., handcrafted) dialogue policy. These systems have a number of "severe weaknesses" such as using unreliable confidence measures, having difficulty coping with the dynamic changing of the user's goal and intention. Moreover, tuning the dialogue policy is labor extensive, based on off-line analysis of the system logs [183].

The first work that applies the POMDP for the dialogue management problem was proposed by *Roy et al.* [142] for building a nursing home robot application. In this application, a flat POMDP model is used where the states represent the *user's intentions*; the observations are the *user's speech utterances*; and the actions are the *system responses*. They showed that the POMDP-based DM copes well with noisy speech utterances, for example their POMDP-based DM makes fewer mistakes than an MDP-based DM, and it automatically adjusts the dialogue policy when the quality of the speech recognition degrades. *Zhang et al.* [187] extended the Roy model in several dimensions: (1) a factored POMDP [22] is deployed for the state and observation sets, (2) the states are composed of the *user's intentions* and *"hidden system states"*, (3) the observations are the *user's utterances* and other observations being inferred from *lower-level information of the speech recognizer, robust parser, and from other input modalities*. *Williams et al.* [178, 179] further extended the Zhang model by adding the state of the dialogue from the perspective of the user which is hidden from the system's view (called *user's dialogue state*) to the state set and adding the confidence score into the observation set. All these approaches have shown that POMDP-based dialogue strategies outperform Markov Decision Process (MDP) counterparts (e.g., *Pietquin* [126]). Furthermore, these strategies cope well with different types of errors in a Spoken Dialogue System (SDS), especially with ASR errors. Table 5.3 describes characteristics of these POMDP-based DMs.

Application	$n, \|S\|, \|A\|, \|Z\|$	Algorithm	Reward function
Nursing home robot [142]	4, 13, 20, 16	AMDP [141]	If the system action is labeled as *correct*: 100, *ok*: -1, *wrong*: -100.
Tour guide [187]	3, 40, 18, 25	QMDP [103], FIB [76]	If the answer matches user's request, the reward is positive. Otherwise, the reward is negative.
Travel booking [174]	2, 36, 5, 5	Perseus [161]	If the system action & dialogue state is *ask* & *not stated*: -1, *ask* & *stated*: -2, *ask* & *confirmed*: -3, *confirmed* & *not stated*: -3, *confirmed* & *stated*: -1, *confirm* & *confirmed*: -2. If the user's goal is determined correctly: 50, incorrectly: -50.

Table 5.1: Characteristics of some POMDP-based dialogue managers (n is the number of slots)

5.4. The factored POMDP approach

We represent our affective dialogue model as a factored POMDP [22]. The state set is composed of the user's goal (Gu), the user's affective state (Eu), the user's action (Au), and the user's grounding state (Du) (similar to the user's dialogue state described in *Williams et al.* [179]). The observation set is composed of the observations of the user's action (OAu) and the observations of the user's affective state (OEu). Depending on the complexity of the application's domain, these features can be represented by more specific features. For example, the user's affective state can be encoded by continuous variables such as *valence* and *arousal*, and can be represented using a continuous-state POMDP [24]. The observation of the user's affective state might be represented by a set of observable effects such as response speech, speech pitch, speech volume, posture, and gesture [13].

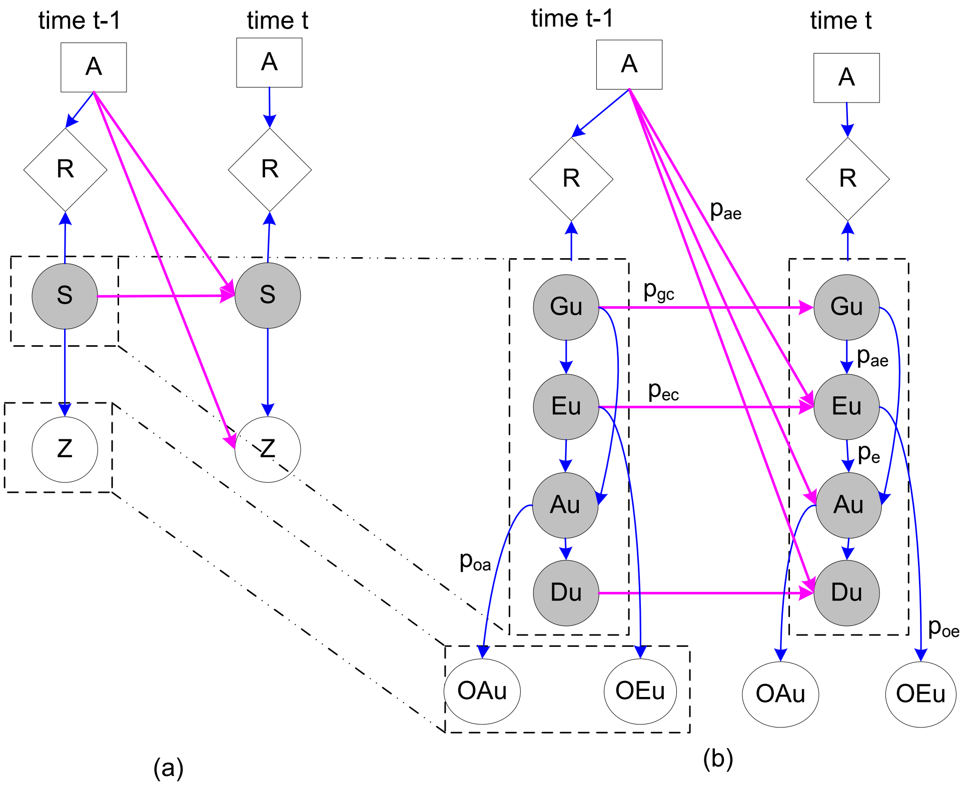


Figure 5.2: (a) Standard POMDP, (b) Two time-slice of factored POMDP for the dialogue manager

Figure 5.2b shows the structure of our affective dialogue model. The features of the state set, action set, observation set, and their dependencies form a two time-slice

Dynamic Bayesian Network (2TBN). Implicitly, some assumptions are made in this model: the user's goal only depends on the user's goal in the previous slice and the system action from the previous slice only influences the user's emotion, the user's action, and the grounding state. We can easily modify this model for representing other dependencies, for example the dependency between the user's emotion and the observation of the user's action. Parameters $p_{gc}, p_{ae}, p_{ec}, p_e, p_{oa}$, and p_{oe} are used to produce handcrafted transition and observation models in case no real data is available (e.g., at the initial phase of the system development), where p_{gc} is the probability of the user goal change; p_{ae} is the probability of the user's emotional change because of the influence of the system action such as when the system confirms an incorrect user's goal (represented by two causal links from the system action and user's goal to the user's affective state); p_{ec} is the probability that the user emotion change is due to the emotion decay and other causes; p_e is the probability of an error in the user's action being induced by emotion; p_{oa} and p_{oe} are the probabilities of the observation error of the user's action and the observation error of the user's affective state, respectively. The reward function is in principle different for each particular application. Therefore it is not specified in our general affective dialogue model.

Suppose the set of user's goals has m values which are represented by $Gu = \{v_1, v_2, ..., v_m\}$. The features of S and Z and the action set A are formulated as follows:

- $Eu = \{neutral, stress, frustration, anger, happiness, ...\}$.
  Note that we can extend the representation of the user's emotion by adding more relevant features into the state space. For example, if the user's emotion is described by two dimensions *valence* and *arousal*. E_u then becomes a sub-network with two continuous variables.

- $Au = \{answer(x), yes, no, ...\}$, where $x \in Gu$.
  The abstract format of Au is $userSpeechAct(x)$, where *userSpeechAct* is an element of the set of the user's speech acts.

- $Du = \{notstated, stated, confirmed, ...\}$.

- $OAu = \{answer(x), yes, no, ...\}$, where $x \in Gu$.
  The value $y \in OAu$ depends on the level of abstraction of the observation of the user's action. For example if the observation of the user's action is sent by the ASR module, y is the word-graph or N-best hypotheses of the user's utterance. In our model, we assume a high level of abstraction for the observation of the user's action such as the output from a dialogue act recognition module or the intention level in the simulated user model [58]. In the latter case OAu has the same set of values as Au.

- $OEu = \{neutral, stress, frustration, anger, happiness, ...\}$.
  Similar to the observation of the user's action, the observation of the user's affective state can be represented by a set of observable effects such as *response speech*, *speech pitch*, *speech volume*, *posture*, and *gesture* features [13]. In our current model, we assume that the observations of the user's affective states are

the output of an AR module, and therefore OEu has the same set of values as Eu.

- $A = \{ask, \mathit{confirm}(x), ...\}$, where $x \in Gu$.
  The abstract format of A is $\mathit{systemSpeechAct}(x)$, where *systemSpeechAct* is an element of the set of the system speech acts.

For a random variable X, we denote x and x' as the values of X at time $t-1$ and t, respectively. Based on the network structure shown in Figure 5.2b, the transition function is represented compactly as follows:

$$\mathcal{P}^a_{ss'} = P(g'_u|g_u)P(e'_u|a, e_u, g'_u)P(a'_u|a, g'_u, e'_u)P(d'_u|a, d_u, a'_u). \tag{5.1}$$

$P(g'_u|g_u)$ is called the the *user's goal model*, $P(e'_u|a, e_u, g'_u)$ is called the *user's emotion model*, $P(a'_u|a, g'_u, e'_u)$ is called the *user's action model*, and $P(d'_u|a, d_u, a'_u)$ is called the *user's grounding state model.* The observation function is as follows:

$$\mathcal{P}^a_{s'z'} = P(\tilde{a}'_u|a'_u)P(\tilde{e}'_u|e'_u), \tag{5.2}$$

where $\tilde{a}'_u \in OAu$ and $\tilde{e}'_u \in OEu$. $P(\tilde{a}'_u|a'_u)$ is called the *observation model of the user's actions* and $P(\tilde{e}'_u|e'_u)$ is called the *observation model of the user's emotions.*

5.5. User simulation

The DM that we have described is a statistical DM. Dialogue corpora are usually used to train this type of DMs. However, the (PO)MDP-based DM has a huge number of states, therefore it is almost impossible to learn an optimal policy directly from a fixed corpus, regardless of its size [147]. To solve this problem, user simulation techniques have been used [58, 98, 126, 148, 149]. The main idea is a two-phase approach. A simulated user is first trained on a small human-computer dialogue corpus to learn responses of a real user given the dialogue context. The learning DM then interacts with this simulated user in a trial and error manner to learn an optimal dialogue strategy. Experimental results show that a competitive dialogue strategy can be learnt even with handcrafted user model parameters [184, 185]. Recent work also demonstrated that user simulation can be used for testing dialogue systems in early phases of the iterative development cycle [3].

Our simulated user model, constructed based on the POMDP environment, is shown in Figure 5.3. The structure of this model is similar to the structure of the POMDP model (Fig. 5.2b), except that the state feature nodes (i.e., Gu, Eu, Au, and Du) in the simulated user model are observable from the user's perspective.

The process to generate observations of the user's actions and of the user's affective states is as follows: First, the value a_{t-1} from the DM is updated on node A of the time-slice $t-1$, the reward r_{t-1} is identified from node A of time-slice $t-1$. Second, the user's goal, affective state, action, and dialogue state are randomly generated based on the probability distributions of the nodes Gu, Eu, Au, and Du (of time-slice t), respectively. Third, the network is updated, and the observation of the user's action oau_t and affective state oeu_t are randomly selected based on the probability distribution of nodes OAu and OEu. The tuple $< r_{t-1}, oau_t, oeu_t >$ is sent back to the DM.

5.6. Single-slot route navigation example

We illustrate our affective dialogue model described in Section 5.4 by a simulated toy route navigation example: "A rescue worker (denoted by "the user") needs to get a route description to evacuate victims from an unsafe tunnel. To achieve this goal, he communicates his current location (one of m locations) to the system. The system can infer the user's stressed state and uses this information to adapt its dialogue policy."

In this simple example, the system can *ask* the user about their current location, *confirm* a location provided by the user, show the route description (*ok*) of a given location, and stop the dialogue (i.e., execute the *fail* action) by connecting the user to a human operator. The factored POMDP model for this example is represented by:

- $S = \langle Gu \times Au \times Eu \times Du \rangle \cup end$, where *end* is an absorbing state.
    1. $Gu = \{v_1, v_2, ..., v_m\}$
    2. $Au = \{answer(v_1), answer(v_2), ..., answer(v_m), yes, no\}$
    3. $Eu = \{e_1, e_2\} = \{nostress, stress\}$
    4. $Du = \{d_1, d_2\} = \{notstated, stated\}$
- $A = \{ask, confirm(v_1), ..., confirm(v_m), ok(v_1), ok(v_2), ..., ok(v_m), fail\}$,
- $Z = \langle OAu \times OEu \rangle$.
    1. $OAu = \{answer(v_1), answer(v_2), ..., answer(v_m), yes, no\}$
    2. $OEu = \{nostress, stress\}$

The full flat-POMDP model is composed of $4m^2 + 8m + 1$ states, $2m + 2$ actions, and $2m + 4$ observations, where m is the number of locations in the tunnel.

We use two criteria to specify the reward model, helping the user to obtain the correct route description as soon as possible and maintaining the dialogue appropriateness [179]. Concretely, if the system *confirms* when the user's grounding state is *notstated*, the reward is -2, the reward is -3 for action *fail*, the reward is 10 when the system gives a correct solution (i.e., the system action is $ok(x)$ where x is the user's goal), otherwise the reward is -10. The reward for any action taken in the absorbing *end* state is 0. The reward for any other actions is -1. Designing a reward model that leads to a good dialogue policy is a challenging task. It requires both expert knowledge and practical debugging (see Appendix A).

The probability distributions of the transition function and observation function are generated using the parameters $p_{gc}, p_{ac}, p_{ec}, p_e, p_{oa}, p_{oe}$ defined in Section 5.4 and two other parameters K_{ask} and $K_{confirm}$, where K_{ask} and $K_{confirm}$ are the coefficients associated with the *ask* and *confirm* actions. We assume that when the user

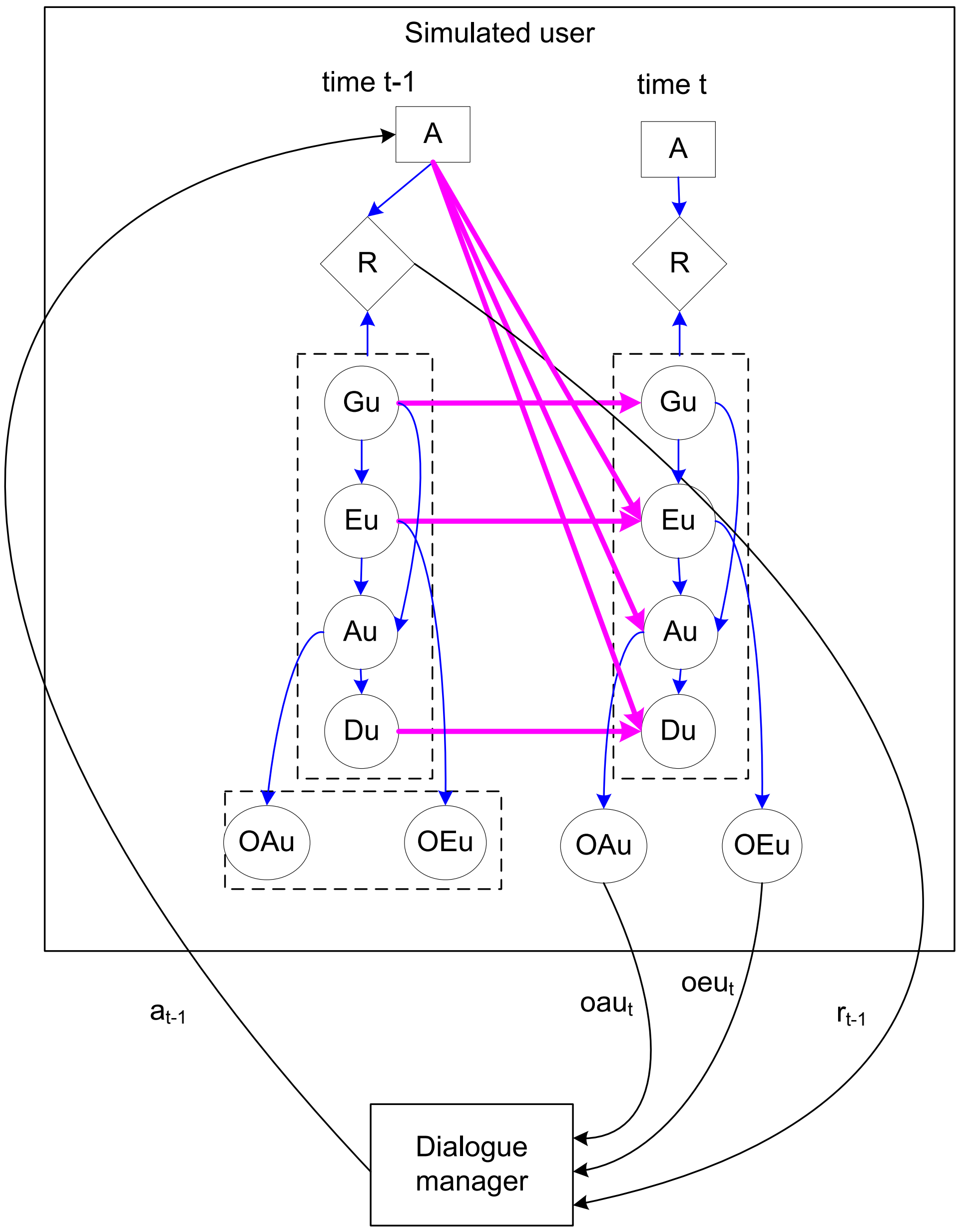


Figure 5.3: Simulated user model using the Dynamic Bayesian Network (DBN). The user's state, action at each time-step are generated from the DBN. Only the observation of the user's action, affective state, and the reward are sent to the dialogue manager.

is stressful, he will make more errors in response to the system *ask* action than the system *confirm* action because in our current model, the number of possible user actions in response to *ask* (`m` possible actions: $answer(v_1), answer(v_2), ..., answer(v_m)$) is greater than to *confirm* (2 actions: *yes*, *no*).

Concretely, the handcrafted models of the transition, observation and reward function are described as follows. The user's goal model is represented by:

$$P(g'_u|g_u) = \begin{cases} 1 - p_{gc} & \text{if } g'_u = g_u, \\ \frac{p_{gc}}{|G_u|-1} & \text{otherwise,} \end{cases} \tag{5.3}$$

where g_u is the user's goal at time $t-1$, g'_u is the user's goal at time t, $|G_u|$ is the number of the user's goals. This model assumes that the user does not change their goal at the next time step with the probability $1 - p_{gc}$.

The user's stress model:

$$P(e'_u|a, e_u, g'_u) = \begin{cases} 1 - p_{ec} & \text{if } e'_u = e_u \text{ and } a \in X, \\ p_{ec} & \text{if } e'_u \neq e_u \text{ and } a \in X, \\ 1 - p_{ec} - p_{ae} & \text{if } e_u = e'_u = e_1 \text{ and } a \notin X, \\ p_{ec} + p_{ae} & \text{if } e_u = e_1 \text{ and } e'_u = e_2 \text{ and } a \notin X, \\ p_{ec} - p_{ae} & \text{if } e_u = e_2 \text{ and } e'_u = e_1 \text{ and } a \notin X, \\ 1 - p_{ec} + p_{ae} & \text{if } e_u = e'_u = e_2 \text{ and } a \notin X, \end{cases} \tag{5.4}$$

where $p_{ec} \geq p_{ae} \geq 0$, $(p_{ec} + p_{ae}) \leq 1$ and $X = \{ask, confirm(g'_u), ok(g'_u)\}$. This model assumes that system mistakes (such as confirming the wrong item) would elevate the user's stress level.

The user's action model:

$$P(a'_u|a, g'_u, e'_u) = \begin{cases} 1 & \text{if } a = a_1,\ e'_u = e_1, \text{ and } a'_u = a_2, \\ 1 - p_1 & \text{if } a = a_1,\ e'_u = e_2, \text{ and } a'_u = a_2, \\ \frac{p_1}{|A_u|-1} & \text{if } a = a_1,\ e'_u = e_2, \text{ and } a'_u \neq a_2, \\ 1 & \text{if } a = a_3,\ e'_u = e_1, \text{ and } a'_u = a_4, \\ 1 - p_2 & \text{if } a = a_3,\ e'_u = e_2, \text{ and } a'_u = a_4, \\ \frac{p_2}{|A_u|-1} & \text{if } a = a_3,\ e'_u = e_2, \text{ and } a'_u \neq a_4, \\ 1 & \text{if } a = a_5,\ e'_u = e_1, \text{ and } a'_u = a_6, \\ 1 - p_2 & \text{if } a = a_5,\ e'_u = e_2, \text{ and } a'_u = a_6, \\ \frac{p_2}{|A_u|-1} & \text{if } a = a_5,\ e'_u = e_2, \text{ and } a'_u \neq a_6, \\ \frac{1}{|A_u|} & \text{if } a = ok(y) \text{ or } a = fail, \\ 0 & \text{otherwise,} \end{cases} \tag{5.5}$$

where $a_1 = ask$, $a_2 = answer(g'_u)$, $a_3 = confirm(g'_u)$, $a_4 = yes$, $a_5 = confirm(x)$, $a_6 = no$, $p_1 = p_e/K_{ask}$, $p_2 = p_e/K_{confirm}$, x & $y \in Gu$, and $x \neq g'_u$.

The main idea behind this handcrafted user's action model is explained as follows. When the user is not stressed, no communicative errors are made. When the user is under stress, they might make errors. The probability that the user makes no communicative errors when the system asks is $1-p_1$ and when the system confirms is $1-p_2$.

The user's grounding state model handcrafted. It is represented by:

$$P(d'_u|a,d_u,a'_u) = \begin{cases} 1 & \text{if } a = ask,\ d_u = d_1,\ a'_u = answer(x), \text{ and } d'_u = d_2, \\ 1 & \text{if } a = ask,\ d_u = d_1,\ a'_u \in \{yes, no\}, \text{ and } d'_u = d_1, \\ 1 & \text{if } a = confirm(x),\ d_u = d_1,\ a'_u = no, \text{ and } d'_u = d_1, \\ 1 & \text{if } a = confirm(x),\ d_u = d_1,\ a'_u \neq no, \text{ and } d'_u = d_2, \\ 1 & \text{if } a \in \{ask, confirm(x)\},\ d_u = d_2 \text{ and } d'_u = d_2, \\ 1 & \text{if } a \in \{ok(x), fail\} \text{ and } d'_u = d_1, \\ 0 & \text{otherwise.} \end{cases} \quad (5.6)$$

The observation model of the user's actions:

$$P(\tilde{a}_u|a_u) = \begin{cases} 1-p_{oa} & \text{if } \tilde{a}'_u = a_u, \\ \frac{p_{oa}}{|A_u|-1} & \text{otherwise,} \end{cases} \quad (5.7)$$

where a_u and $\tilde{a}'_u$ are the user's action and the observation of the user's action, respectively. $|A_u|$ is the number of the user's actions. Parameter p_{oa} will be the concept error rate if we consider each user's action is a distinctive concept.

The observation model of the user's stress:

$$P(\tilde{e}_u|e_u) = \begin{cases} 1-p_{oe} & \text{if } \tilde{e}_u = e_u, \\ \frac{p_{oe}}{|E_u|-1} & \text{otherwise,} \end{cases} \quad (5.8)$$

where e_u and $\tilde{e}_u$ are the user's stress state and the observation of the user's stress state, respectively. $|E_u|$ is the number of the user's stress states. Parameter p_{oe} can be considered as the recognition error rate of an affect recognition module used for the user's stress detection.

The reward function:

$$R(s,a) = \begin{cases} 0 & \text{if } s = end, \\ -2 & \text{if } a = ask \text{ and } g_u = stated, \\ -2 & \text{if } a = confirm(x) \text{ and } g_u = notstated, \\ 10 & \text{if } a = ok(x) \text{ and } g_u = x, \\ -10 & \text{if } a = ok(x) \text{ and } g_u \neq x, \\ -3 & \text{if } a = fail, \\ -1 & \text{otherwise.} \end{cases} \quad (5.9)$$

5.7. Evaluation

To compute a near-optimal policy for the route navigation example presented in Section 5.6 we use two state-of-the-art approximate POMDP solvers: Perseus[2] and

[2]http://staff.science.uva.nl/ mtjspaan/software/approx/[accessed 2008-05-14]

ZMDP[3]. These solvers are implemented based on the Perseus and HSVI2 algorithms presented in Chapter 2. All experiments were conducted on a Linux server using a 3 GHz Intel Xeon CPU and a 24 GB RAM.

The performance of the computed policy is then evaluated using the simulated user presented in Section 5.5. The discount factor is only used for the planning phase. In the evaluation phase the total reward for each dialogue session is the sum of all the rewards the system receives at each turn during the system-user interaction process (i.e., $\gamma = 1$). There are two reasons for this decision. First, in the dialogue domain, the number of turns in a dialogue session between the system and the user is finite. Second, the intuitive meaning of the discount factor ($\gamma < 1$) is reasonable for positive reward values but is not appropriate for negative reward values. For example, it is less likely that the second *confirm* action bears a smaller cost than the first one on a given piece of information provided by the user.

A dialogue session between the system and the user is defined as an *episode*. Each episode in the route navigation application starts with the system's action. Following the turn-taking mechanism, the system and the user exchange turns[4] until the system selects an *ok* or *fail* action (Table 5.2). The episode is then terminated[5]. Formally, the reward of each episode is calculated from the user side as follows

$$R_e = \sum_{t=0}^{n} R_t(S_t, A_t), \tag{5.10}$$

where n is the number of turns of the episode, the reward at turn t $R_t(S_t, A_t)$ is equal to R_s^a (see Section 2.5) if the user's state and action at turn t is $S_t = s$ and the system action at turn t is a. Note that each turn is composed of a pair of system and user's actions except the last turn. Table 5.2 shows an example of a 3-turns episode of the single-slot route navigation application.

Turn		Utterance	Reward
1	S_1 :	Please provide the location of the victims? [*ask*]	-1
	U_1 :	Building 1. [$answer(v_1)$]	
2	S_2 :	The victims are in the Building 1. Is that correct? [$confirm(v_1)$]	-1
	U_2 :	Yes. [*yes*]	
3	S_3 :	Ok, the route description is shown on your PDA. [$ok(v_1)$]	10
		Reward of the episode:	8

Table 5.2: An episode of the interaction between the system and the user

[3]http://www.cs.cmu.edu/ trey/zmdp/[accessed 2008-05-14]

[4]Each time step in a formal POMDP definition in Chapter 2 is now considered as a turn.

[5]Appendix A shows a telephone-based dialogue example where the episode is ended when the user hangs up.

The average return of N episodes is defined as follows:

$$R_N = \frac{1}{N}\sum_{e=1}^{N} R_e \tag{5.11}$$

In the following sections, we first present the experiments for tuning three important parameters: the discount factor, the number of belief points and the planning time. Second, we show the influence of the stress to the performance of computed policies. Third, we compare the performance of the approximate POMDP policies versus three handcrafted policies and the greedy action selection policy. We then conduct experiments to address tractable issues.

5.7.1. Parameter tuning

Figures 5.4, 5.5, and 5.6 show the average returns of the simplest instance of the route navigation problem (3 locations) where the near-optimal policy is computed based on different values of the discount factor, the number of belief points (only for Perseus), and the run-time for the planning phase.

Based on the result shown in Figure 5.4, the discount factor value $\gamma = 0.99$ is selected for subsequent experiments that will be presented in this chapter. Interestingly, it turns out that the *de facto* discount factor value ($\gamma = 0.95$) that is usually used for POMDP-based dialogue problems from the literature (e.g., *Williams and Young* [177]) is not a good solution, at least for this example. It is worth noting that the off-line planning time to get an ϵ-optimal policy increases monotonically with the value of the discount factor especially when the convergence threshold ϵ is small. Perseus converges to a stable solution quicker than ZMDP[6]. For example, the time to get an ϵ-optimal policy ($\epsilon = 0.001$) computed using the Perseus solver for γ is equal to 0.9, 0.95, and 0.99 is 7, 17, and 98 seconds, respectively. Meanwhile, the time to get a similar policy computed using the ZMDP solver for γ is equal to 0.9, 0.95, and 0.99 is 3, 53, and $> 10^6$ seconds. However, the time-bounded solution (Fig. 5.4) computed by both solvers performs well when we conduct the test with our simulated user.

Figure 5.5 shows that a reasonable solution is achieved with the number of belief points starting from 200. Given a fixed threshold of the planning time, the number of iterations decreases when the number of belief points increases. That is why the average return of the case of 10000 belief points is low when the planning time is limited to 60 seconds. A default number of belief points for all subsequent experiments is set to 1000.

Figure 5.6 shows that a stable solution is achieved for both Perseus and ZMDP solvers when the planning time threshold is 60 seconds. ZMDP converges to a good solution quicker than Perseus (0.5 seconds vs. 60 seconds). For all the subsequent experiments for the 3-locations case the default threshold for the planning time is

[6]This comparison is only relative based on our empirical test. The convergence threshold of ZMDP is the difference between the upper bound and lower bound value functions. On the other hand, the convergence threshold of Perseus is the difference between the current value function and the previous one. It is obvious that the convergence threshold of ZMDP is closer than that of Perseus.

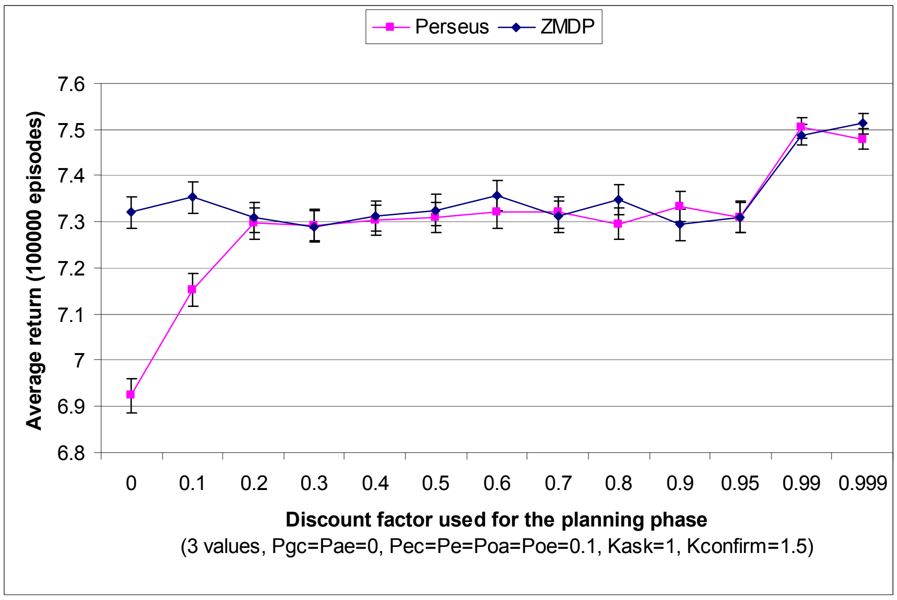


Figure 5.4: Average return vs. the discount factor used for the planning phase. Error bars show the 95% confidence level. The threshold of the planning time is 60 seconds. Policies with $\gamma \leq 0.95$ converge ($\epsilon = 0.001$) before this threshold.

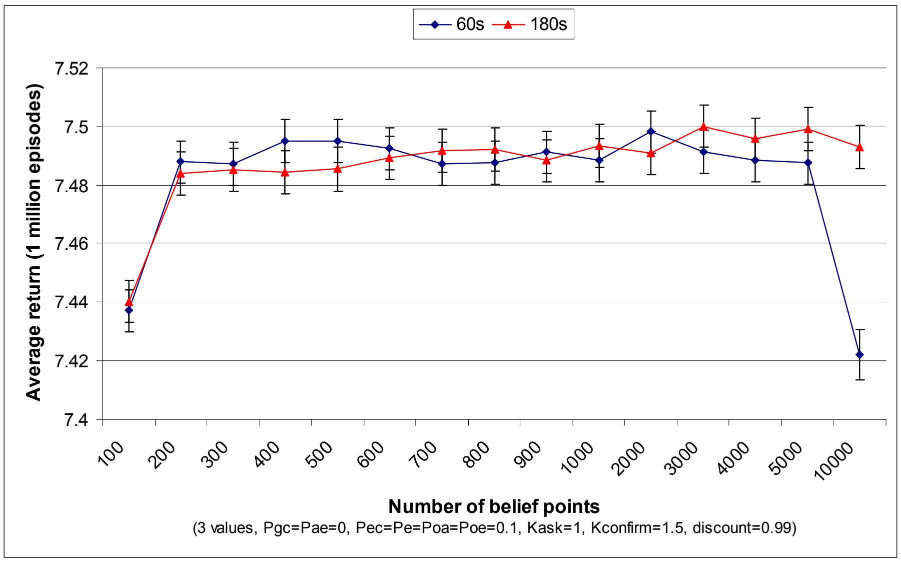


Figure 5.5: Average return vs. number of belief points. Error bars show the 95% confidence level.

set to 60 seconds. When the number of locations increases both Perseus and ZMDP need a longer time to get a good solution. For example, for a 10 locations case, the minimum time for Perseus is 30 minutes and for ZMDP is 10 minutes.

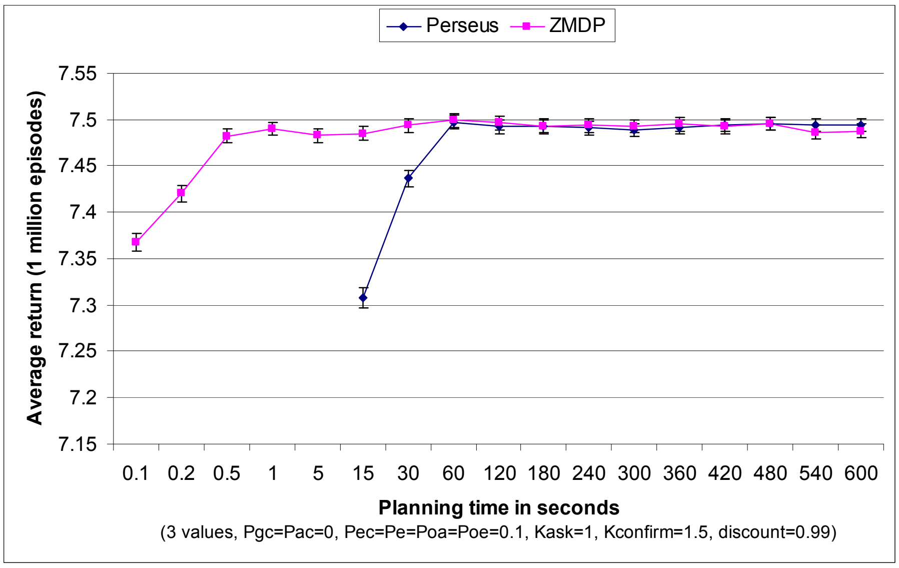


Figure 5.6: Average return vs. planning time in seconds. Error bars show the 95% confidence level.

5.7.2. Influence of stress to the performance

Figure 5.7 shows the influence of stress to the performance of two distinctive policies computed using Perseus: the non-affective policy (SDS-POMDP) and the affective policy (ADS-POMDP). The SDS-POMDP does not incorporate the stress variable in the state and the observations set (similar to the SDS-POMDP policy described by *Williams and Young* [177]). The ADS-POMDP is our affective dialogue model described in Section 5.4. The probability of the user's action error being induced by stress p_e changes from 0 (stress has no influence to the user's action selection) to 0.8 (the user is highly stressed and acts almost randomly). The average returns of both policies decreases when p_e increases. When stress has no influence on the user's action error, the average returns of the two policies are equal. When $p_e \geq 0.1$, the ADS-POMDP policy outperforms the SDS-POMDP counterpart[7].

[7]We then assume that the ADS-POMDP policy is better than the MDP policy since *Williams* [175] demonstrated that the SDS-POMDP policy outperforms its MDP counterpart.

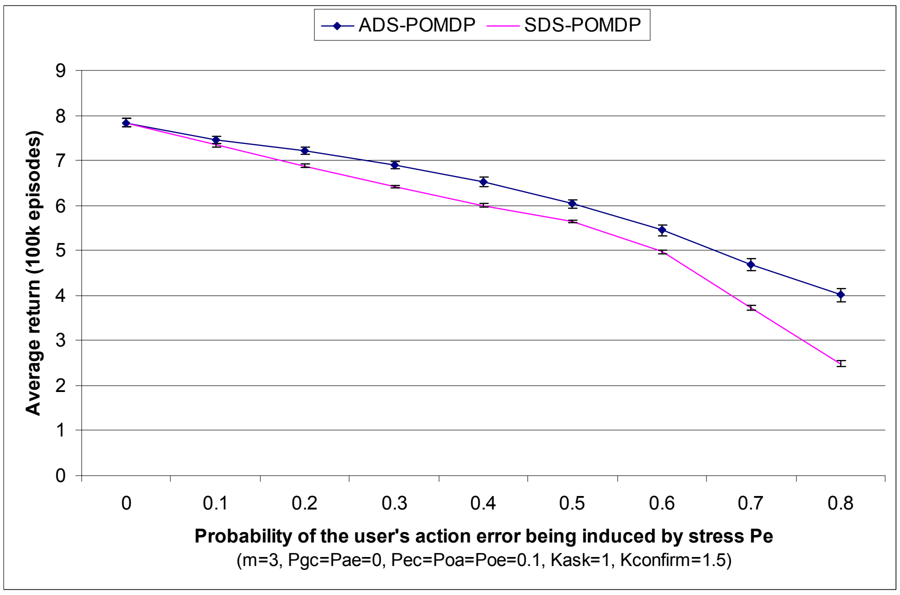


Figure 5.7: Average returns of the affective policy and non-affective policy vs. the probability of the user's action error induced by stress p_e

5.7.3. Comparison with other techniques

In this section, we evaluate the performance of the POMDP DM by comparing the performance of the approximate POMDP policy computed using Perseus (ADS-POMDP) and four other dialogue strategies: HC1, HC2, HC3 (Fig. 5.8), and the greedy action selection strategy. HC1 is the optimal dialogue policy when $p_{gc} = p_e = p_{oa} = 0$ (the user's goal does not change; stress has no influence on the user's action and there is no error in observing the user's action i.e., the speech recognition and spoken language understanding errors are equal to 0). HC1 and HC2 are considered as the non-affective dialogue strategies since they ignore the user's stress state. HC3 uses commonsense rules to generate the system behavior. The greedy policy is a special case of the POMDP-based dialogue with the discount factor $\gamma = 0$ (this strategy is similar to the one used in two real-world dialogue systems [118, 185]).

As expected, the ADS-POMDP policy outperforms all other strategies (Fig. 5.9). HC3 outperforms its handcrafted counterparts

5.7.4. Tractability

The ϵ-optimal policy presented in Sections 5.7.1, 5.7.2, and 5.7.3 is computed for the simplest instance of the single-slot route navigation example which is composed of only three locations ($m = 3$). In reality, even with a single-slot dialogue problem, the number of slot values m is usually large. Section 5.6 presents a connection between

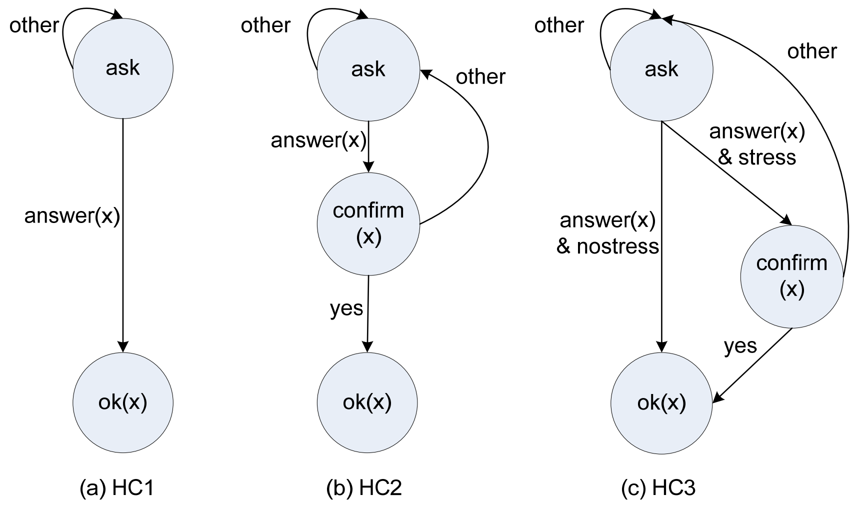


Figure 5.8: Three handcrafted dialogue strategies for the single-slot route navigation problem (x is the observed location): (a) first *ask* and then select *ok* action if the observation of the user's action $\tilde{a}_u$ is *answer* (otherwise *ask*), (b) first *ask*, then *confirm* if $\tilde{a}_u = answer$ (otherwise *ask*) and then select *ok* action if $\tilde{a}_u = yes$ (otherwise *ask*), (c) first *ask*, then *confirm* if $\tilde{a}_u = answer$ & $\tilde{e}_u = stress$ and select *ok* action if $\tilde{a}_u = yes$.

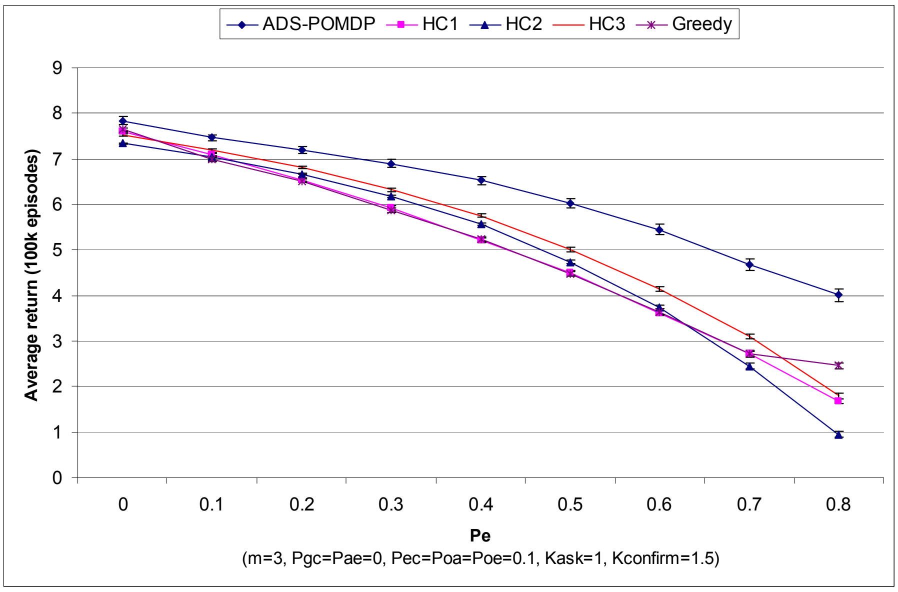


Figure 5.9: Average return of the POMDP policy vs. other policies

m and the size of the POMDP problem (state, action, and observation sets). For the single-slot route navigation example, the number of states is a quadric function of m. The numbers of actions and observations are also a linear function of m. In this section, we address the POMDP tractable issues by increasing the number of slot values gradually and trying to compute the ϵ-optimal policy for each case.

Figure 5.10 shows that a portion of the planning time of the ZMDP solver (time after first call to the solver) increases exponentially in the dimension of m. A reasonable policy can only be obtained with $m \leq 45$. Perseus scales worse than ZMDP. It can only handle the problem with $m \leq 15$.

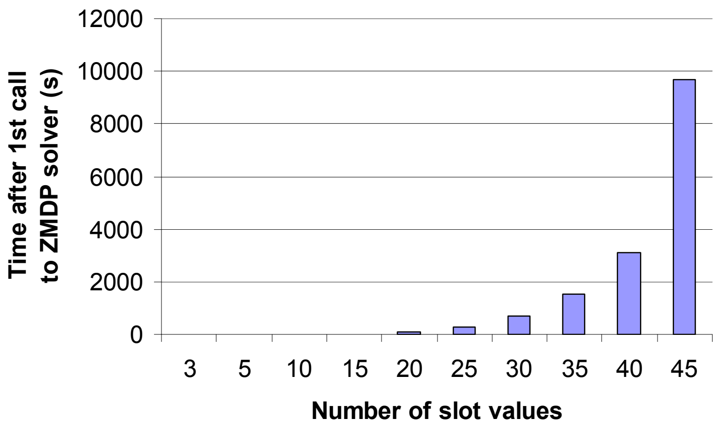


Figure 5.10: Planning time vs. number of slot values

This is because although the approximate PBVI algorithms such as Perseus or HSVI2 are able to handle the curse of history problem, the curse of dimensionality (i.e., the dimensionality of α-vectors grows exponentially with the number of states) remains. Another practical issue is that the size of the optimized POMDP parameter file also increases exponentially in the dimension of m. A recently implemented POMDP solver, Symbolic Perseus, allows for a compact representation of the POMDP parameter files. Symbolic Perseus can help to scale the range of solvable problems to two orders of magnitude compared with the ZMDP solver. As described in Chapter 2, it was demonstrated to solve a hand-washing problem with the size of 50 million states, 20 actions, and 12 observations. Although this is one of the most complex problems in the POMDP research community, it is only a toy problem compared with real-world dialogue problems. For example, the affective dialogue model for the RestInfo problem presented in Chapter 2 is composed of more than 600 million states and its spoken counterpart is composed of more than 300 million states.

5.8. Conclusions

We have presented a factored POMDP approach to affective dialogue modeling and illustrated our affective dialogue model through a single-slot route navigation example. The 2TBN representation allows integration of the features of states, actions, and observations in a flexible way. We have also shown that even if the observation is perfect, the expected return of the optimal dialogue strategy depends on the correlation between the user's affective state and the user's action. The POMDP dialogue strategy outperforms four other strategies (three handcrafted and greedy action selection strategies). Furthermore, the POMDP dialogue strategy copes well with different types of errors such as the speech recognition error[8] and the user's action error being induced by stress as shown in Section 5.7.2. However, solving the POMDP problem (i.e., finding a near-optimal policy) is computationally expensive. Therefore, all currently developed POMDP dialogue management work is limited to toy frame-based dialogue problems with the size of several slots (e.g., two slots in *Williams and Young* [174], three slots in *Zhang et al.* [187], and four slots in *Roy et al.* [142]) (Table 5.3). Chapter 6 proposes a method to overcome this limitation.

[8]See the evaluation of two other dialogue problems reported in Appendix A

Chapter 6

Scaling up: The DDN-POMDP approach

Part of this chapter is presented in Bui et al. [32, 34]. The DDN-POMDP approach described in this chapter is an important contribution of the thesis.

6.1. Introduction

In Chapter 5 we showed that Partially Observable Markov Decision Processs (POMDPs) are an elegant model for designing Affective Dialogue Systems (ADSs). We also analyzed the limitation of the current state-of-the-art approximate POMDP algorithms. This approach, therefore, is only applicable for small-scale dialogue problems with the size of only a few slots and slot values (see Table 5.3). Recently, *Williams and Young* [176] proposed a scaled-up POMDP method, called CSPBVI, to deal with the multi-slot problem. The Dialogue Manager (DM) is decomposed into two POMDP levels, a master POMDP and a set of summary POMDPs. Each summary POMDP corresponds to a slot. The optimization or planning step of this method is conducted off-line by simplifying the user behavior (assuming that when the users are asked about a certain slot, they only provide a value for that slot) and reducing the size of the POMDP structure (e.g., approximating the number of values of each slot by only two values *best* and *rest*). However, the affective dialogue model requires a more complex POMDP structure than that of the spoken counterpart. Compressing the POMDP structure prevents us to incorporate a rich model of the user's affect into the state space and might loose dependencies between the user's emotion, goal, and other hidden state variables.

Looking beyond the POMDP framework, another technique to deal with partially observable situations is Dynamic Decision Networks (DDNs). A DDN is an extension of the Dynamic Bayesian Network (DBN) with decision and utility nodes. DDNs provide a concise representation for large POMDPs and can be used as inputs for any POMDP algorithm such as Value Iteration (VI) and policy-iteration based algorithms [145, chap. 17]. A special case of the DDNs (selecting actions based on immediate reward) was used to model real-world dialogue management systems [118, 185]. DBNs and DDNs are also suitable for use in developing the affective user model [56, 100, 191].

In this chapter, we propose a novel method to handle multi-slot dialogue problems without compressing the POMDP structure. Our approach focuses on a real-time belief update and an on-line action selection for a general probabilistic frame-based (or slot-filling) Dialogue System (DS)[1]. The term "probabilistic frame-based" is used because, instead of keeping track of the slot values provided by the user, the DM maintains their probability distributions. Each slot is first formulated as a POMDP and then approximated by a set of DDNs [90]. The approach is, therefore, called the DDN-POMDP approach. It has two new features: (1) being able to deal with a large number of slots (with a large number of slot values) and (2) being able to take into account some aspects of the user's affective state in deriving the adaptive dialogue strategies.

In the following, we first describe a general affective dialogue model using the DDN-POMDP approach. We then present a simulated multi-slot route navigation example and an evaluation of our method. Finally, we summarize the proposed approach.

6.2. The DDN-POMDP approach

The main idea of the DDN-POMDP approach is to split the DM into two levels: (1) the slot level DM and (2) the global DM. The first part is composed of a set of `n` slots $f_1, f_2, ..., f_n$ where each slot f_i is formulated as a POMDP (called the slot-POMDP and denoted by SP_i). The second part, the global DM, is handcrafted. The global DM aims to keep track of the current dialogue information state and to aggregate the system slot actions nominated by the slot-POMDPs. These two parts and the DM activity process are explained in detail in the next sections.

6.2.1. Slot level dialogue manager

We represent each slot as a *factored* POMDP [22] where the state set and observation set are composed of six features similar to the one presented in Section 5.4. The state set is composed of the *user's goals for the slot i* (Gu_i), the *user's affective states* (Eu), *the user's actions for the slot i* (Au_i), and the *user's grounding states for the slot i* (Du_i). The observation set is composed of the *observations of the user's actions for the slot i* (OAu_i) and the *observations of the user's affective states* (OEu). Two of these six features (Eu and OEu) are identical for all slots. The action set defines the *system actions for slot i* (A_i).

[1]When designing a frame-based DS, the application is usually formulated by a set of frames, each frame is composed of a set of relevant slots. The set of frames can easily convert to a set of slots by using standard database normalization procedures (see Chapter 3).

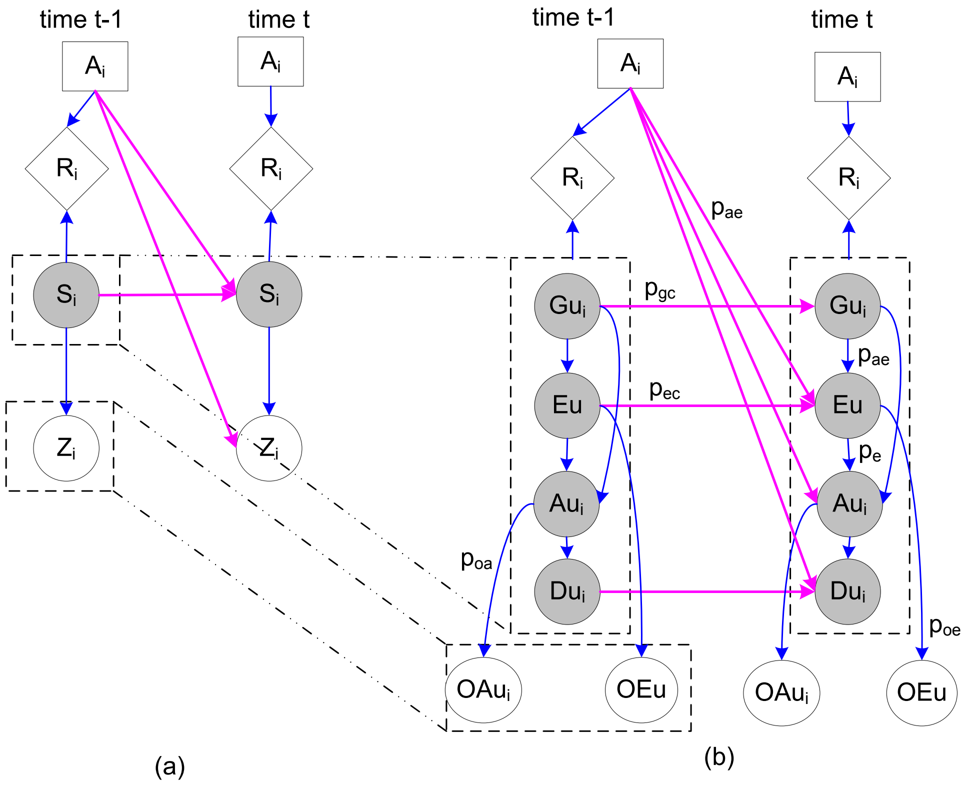


Figure 6.1: (a) Standard POMDP, (b) Two time-slice of the factored POMDP for slot i, where state set S is factored into four features Gu_i, Eu, Au_i, and Du_i, observation set Z is factored into two features OAu_i and OEu. This figure is similar to Fig. 5.2.

Let us consider a simple example: "A rescue worker needs to get a route description to evacuate victims from an unsafe tunnel. To achieve this goal, he communicates the location of the victims (one of three locations) to the system. The system can infer the user's stress state and uses this information to adapt its dialogue policy[2]".

The factored POMDP model SP_i for this example is represented by:

- $S = \langle Gu_i \times Au_i \times Eu \times Du_i \rangle \cup end$, where *end* is an absorbing state.
    1. $Gu_i = \{v_1, v_2, v_3\}$
    2. $Au_i = \{answer(v_1), answer(v_2), answer(v_3), yes, no\}$
    3. $Eu = \{nostress, stress\}$
    4. $Du_i = \{notstated, stated\}$
- $A = \{ask, confirm(v_1), confirm(v_2), confirm(v_3), ok(v_1), ok(v_2), ok(v_3), fail\}$,
- $Z = \langle OAu_i \times OEu \rangle$.
    1. $OAu_i = \{answer(v_1), answer(v_2), answer(v_3), yes, no\}$
    2. $OEu = \{nostress, stress\}$

We are interested in finding an optimal dialogue policy for our POMDP-based DSs. One intuitive approach (as presented in Chapter 5) is to compute a near-optimal dialogue policy using a good approximate POMDP algorithm and to use the result, usually in the form of a policy graph or value function (see Section 2.5), for selecting the appropriate system action. Exact POMDP algorithms such as the Witness [40] cannot be applied even for the above slot-POMDP because of the computational intractability. However, a good approximate algorithm such as Perseus is only tractable for a small number of slot values (≤ 15).

Therefore, to maintain the tractability and allow the real-time belief update for applications having a large number of slot values, we approximate each slot-POMDP by a k-step look-ahead DDN (denoted by kDDN, $k \geq 0$). Conceptually, this approximation gains the tractability by limiting the number of look-ahead steps (an infinite-horizon POMDP is equivalent to a kDDN with $k = \infty$). The kDDN has $(k + 2)$ slices. The first two slices are similar to the 2TBN shown in Figure 6.1b, the next k slices are used to predict the user behavior in order to allow the DM to select the appropriate system action. Because the DM only needs to update its current belief state relevant to the system's last action, each kDDN (of slot i) can be converted to a set of DDNs (called action DDNs and denoted by kDDNAs), one for each action for slot i to simplify the computability in implementing the system. Figure 6.2 shows a structure of the kDDN and kDDNA ($k = 1$) used for SP_i of our route navigation example (Section 6.3). The connections from the action nodes to immediate reward nodes in the next slices are used to model that when a system slot action is selected that leads to the absorbing *end* state (such as *ok* or *fail*), the reward in all next slices are equal to zero[3].

[2]This example is an instance of the single-slot route navigation example described in Section 5.6.

[3]An alternative solution is to add *end* nodes into the kDDNA (one for each slice) and redirect the connections from the action nodes to the immediate reward nodes to these end nodes.

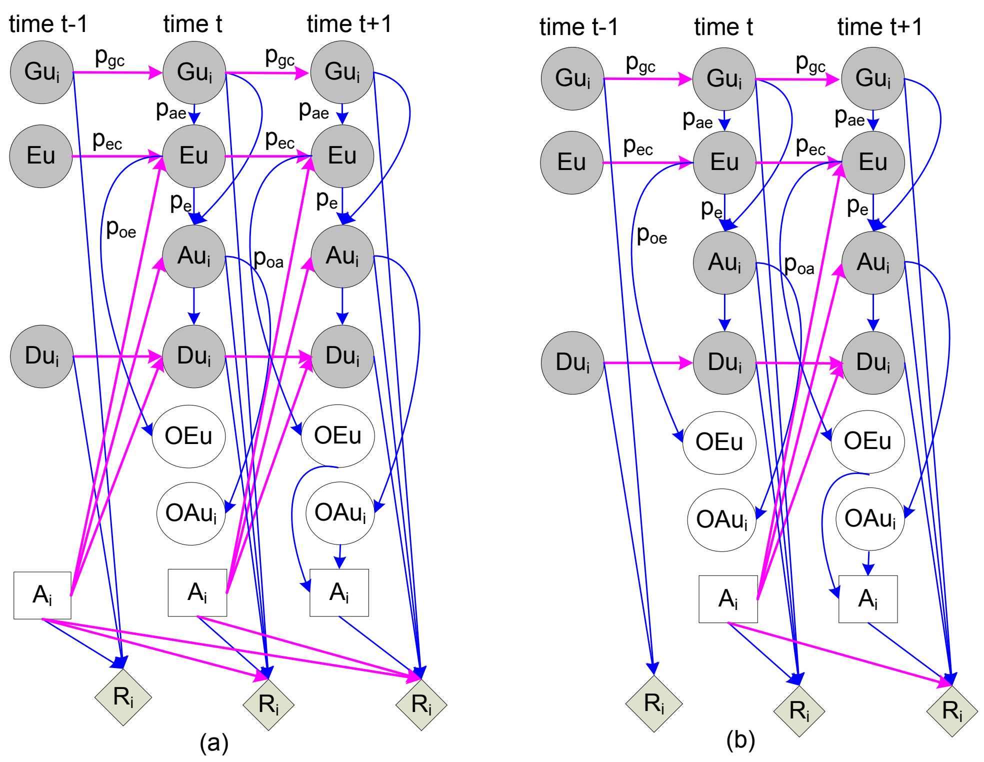


Figure 6.2: The structure of (a) kDDN and (b) kDDNA with one-step look-ahead (i.e., $k = 1$). Shaded-round nodes are hidden, clear-round nodes are observable, rectangular nodes are decision nodes, diamond shaped nodes are reward nodes. Both networks have similar structures except the kDDNA does not have the action node in the first slice. In our implemented prototype DM we use the simpler network kDDNA (to reduce the computation time for the belief update process) directly because the DM can keep track of the last system action, therefore it can update directly on the relevant kDDNA instead of kDDN.

6.2.2. Global dialogue manager

The global DM is composed of two components: the Dialogue Information State (DIS) and the Action Selector (AS). The DIS component is considered as the active memory of the DM. It updates and memorizes the current probability distributions of the user's goal, the user's affective state, the user's action, the user's grounding state of all slots, the previous system action, and the recent observation of the user's action and affective state. The DIS is formally defined by a tuple $\langle \mathbf{P}(Gu), P(Eu), \mathbf{P}(Au), \mathbf{P}(Du), hlsa, a, oau, oeu \rangle$, where:

- $\mathbf{P}(Gu)$ is an `n`-tuple, of which each element i represents the current probability distribution of the user's goal for the slot i. Similarly, $\mathbf{P}(Au)$ and $\mathbf{P}(Du)$ are `n`-tuples, of which each element i represents the current probability distribution of the user's action and the user's grounding state for the slot i, respectively;

- $P(Eu)$ is the probability distribution of the user's affective state;

- *hlsa* is an `n`-tuple, of which each element i is the most recent local action nominated by slot i. The tuple *hlsa* is used by the AS to determine the global system action;

- $a \in A$ is the previous system action, where A is described in Equation 6.2;

- $oau \in OAu$ and $oeu \in OEu$ are the recent observations of the user's action and affective state. Let $F = \{f_1, f_2, ..., f_n\}$ be the set of slots, the observation set of the user's action is represented by:

  $$OAu = \{userSpeechAct(I)\}, \tag{6.1}$$

  where $I \subset F \cup \{(f = x) | f \in F, x \in Gu_i\}$, Eu and OEu are defined in the previous chapter (Section 5.4).

Table 6.1 shows a snapshot of the DIS component of a 2-slot DS taken from our current prototype DM. The first slot has two values, the second slot has three values. The previous system action is *askopen*. The current observation is $oau = answer(f_1 = v_1)$ and $oeu = no$. Only the probability distributions of the first slot are updated based on the observation of the user action.

The DM selects relevant kDDNAs based on the last system action a and the recent observation of the user's action *oau*. When a kDDNA for the slot i is selected, the element i of tuples $\mathbf{P}(Gu)$, $\mathbf{P}(Au)$, and $\mathbf{P}(Du)$ are used to initialize the kDDNAs.

The AS component is responsible for aggregating the system's slot actions nominated by slot-POMDPs. The system action set generated by the AS component is represented by:

$$A = \{systemSpeechAct(I), giveSolution(J), stop\}, \tag{6.2}$$

where $I \subset F \cup J$ and $J = \{(f = x) | f \in F, x \in Gu_i\}$. *giveSolution* and *stop* are two special system actions that lead to the absorbing *end* state. The *giveSolution*

Current user's goal $\mathbf{P}(Gu) = \langle P(Gu_1), P(Gu_2)\rangle$	$P(Gu_1) = (0.921\ 0.079)$ $P(Gu_2) = (0.333\ 0.333\ 0.333)$
Current user's emotion	$P(Eu) = (0.917\ 0.025\ 0.023\ 0.022\ 0.013)$
Current user's action $\mathbf{P}(Au) = \langle P(Au_1), P(Au_2)\rangle$	$P(Au_1) = (0.958\ 0.035\ 0.003\ 0.003)$ $P(Au_2) = (0.2\ 0.2\ 0.2\ 0.2\ 0.2)$
Current user's dialogue state $\mathbf{P}(Du) = \langle P(Du_1), P(Du_2)\rangle$	$P(Du_1) = (0.006\ 0.994)$ $P(Du_2) = (1.000\ 0.000)$
Current observation of the user action	$oau = answer(f_1 = v_1)$
Current observation of the user emotion	$oeu = no$
History of previous system slot actions	$hlsa = \langle ask\ ask\rangle$
Previous system action	$a = askopen$

Table 6.1: An example of the DIS for a 2-slot case, slot f_1 has 2 values, slot f_2 has 3 values.

action aims to provide a solution to the user's request. The *stop* action terminates the current dialogue session.

The AS is heuristic and application-dependent. An example of a set of rules to select a global system action is described in Section 6.3.2.

The idea of splitting the DM into two levels was inspired by the Rapid Dialogue Prototyping Methodology (RDPM) (see Chapter 3). The factored POMDP model for each slot is extended from Williams's model [179]. Two recent dialogue management frameworks using DDNs [118, 185] are comparable to the special case of our method with zero-step look-ahead DDN ($k = 0$). The DDN-POMDP dialogue manager currently does not use a confidence score. However, it is straightforward to incorporate this feature by adding a confidence score variable into the observation set similar to the one used in Williams's model.

6.2.3. Dialogue manager activity process

When the DM is initialized, it loads `n` slot-POMDP parameter files and creates a set of kDDNAs ($|A_i|$ kDDNAs are created from the slot-POMDP parameter file i). The entire process of the DM is explained in this section by a cycle of four steps (Fig. 6.3).

- **Step 1**
  When the DM starts, the kDDNAs nominate greedy actions (i.e., actions associated with highest immediate utilities) to the global DM based on the set of prior probability distributions specified in the slot-POMDP parameter files. These actions are combined by the AS. The output is sent to the user (through the output generation module).

- **Step 2**
  The global DM then receives the observation of the user's action and user's affective state ($oau \in OAu$ and $oeu \in OEu$) (sub-step 2.1). The kDDNAs relevant to oau are activated to compute the next slot action (sub-step 2.2).

The relevant previous belief and current observation (taken from Z and S) are then updated to these kDDNAs (sub-step 2.3). The result computed from the kDDNAs (i.e., the updated probability distributions of the state set, see Fig. 6.3) is propagated back to the DIS component (sub-step 2.4).

- **Step 3**
  All new actions computed from the selected kDDNAs are updated to variable *hsla*. The AS then produces a new system action.

- **Step 4**
  The process repeats from step 2 until the global DM selects either *giveSolution* or *stop* action.

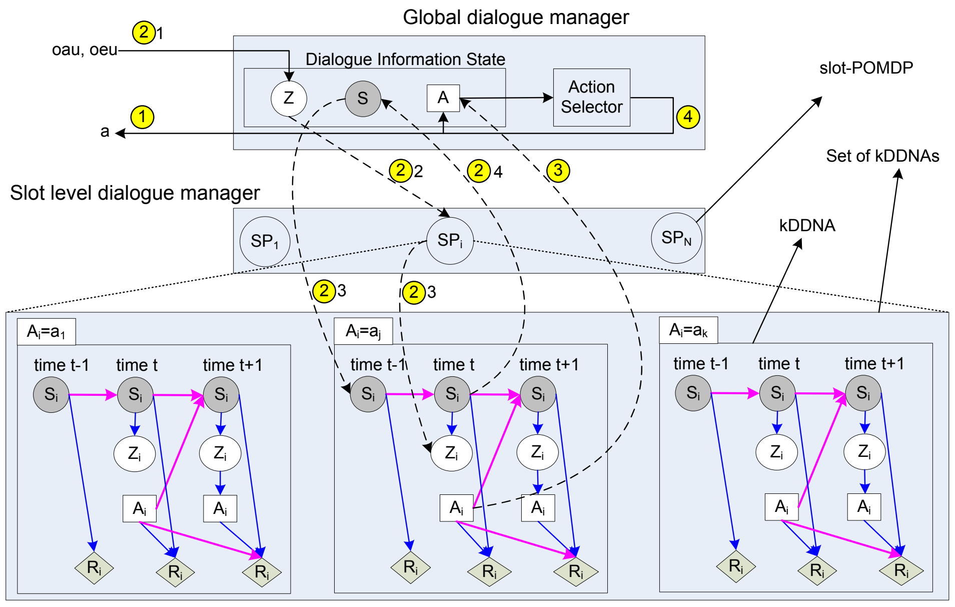


Figure 6.3: Activity process of the DM. The DIS component is represented by three nodes Z, S, and A. Node Z is composed of *oau* and *oeu*. Node S is composed of $\mathbf{P}(Gu), P(Eu), \mathbf{P}(Au), \mathbf{P}(Du)$. Node A is composed of *hlsa* and *a*.

6.2.4. User simulation

The user simulation model presented in Section 5.5 only works for a single-slot dialogue problem. For a multi-slot dialogue problem, the simulated user needs to decide (probabilistically) which slots are going to be selected given the system action. This section presents an extension of the single-slot user simulation model (Sec. 5.5) for the multi-slot dialogue problems. The multi-slot user simulation (Fig. 6.4) is composed of a *slot selection* module, an *observation generation* module, a *global reward function* module, and an `n`-*tuple of DBNs* (one for each slot). The slot selection module determines which slots will be included into the user's response based on the system action and a slot probability distribution P_{sd}. P_{sd} can be estimated from a dialogue corpus as illustrated in Section 6.4.3. The observation generation and global reward function modules simply combine observations and rewards from the selected slots, respectively. Note that it is sufficient to generate the observation of the user's affective state from the first selected slot. The structure of each DBN (Fig. 6.4) is similar to the structure of the POMDP model for each slot (Fig. 6.1b), except that the state feature nodes (i.e. Gu_i, Eu, Au_i, and Du_i) in the simulated user model are observable from the user's perspective.

The process to generate observation of the user's actions and affective states is as follows. First, the value a from the DM is sent to the slot selection module. This module randomly selects a subset of slots being included in the (user's) response based on the probability distribution P_{sd}. If no slot is selected, then the user's action is *null*. Second, the slot action extracted from the DM action a is then updated on node A_i of the time-slice $t-1$, the reward r_1 is identified from node A_i of time-slice $t-1$. Third, the user's goal, affective state, action, and dialogue state are randomly generated based on the probability distribution of the nodes Gu_i, E_u, Au_i, Du_i (of time-slice t), respectively. Fourth, the network is updated, and the observation of the user's action oau'_1 and affective state oeu' are randomly selected based on the probability distribution of nodes OAu_i and OEu. The observations and the rewards are then sent to the observation generation and the global reward function modules. Other selected slots are processed in the same manner as the first one. Finally, the tuple $< r, oau', oeu' >$ from the observation generation and the global reward function modules is sent back to the DM.

Our simulated user model is goal-oriented. The user's action is consistent with the user's goal except when the user is stressed, then they might make a communicative error. The technique used is similar to the Pietquin model which is one of the best simulated user models [92, 147]. *Schatzmann et al.* [147] showed that a strategy learnt with a good user model still performs well when tested on a different user model. Therefore, we expect that our dialogue policy will still be robust when tested with different user simulators.

6.3. Multi-slot route navigation example

The hypothetical scenario example to illustrate our proposed DDN-POMDP approach is as follows: "A serious accident has happened in the Benelux tunnel, Rotterdam, The Netherlands. A rescue team is sent to the tunnel to evacuate a large number of victims. The rescue members are currently at `n` different locations in and around the tunnel. Each location might have one or more rescuers. The team leader (denoted by "*the user*") needs a route description to coordinate with all other team members. He communicates with the DS to get this information. The system is able to produce the route description when it knows the `n` locations of the rescue members. The current

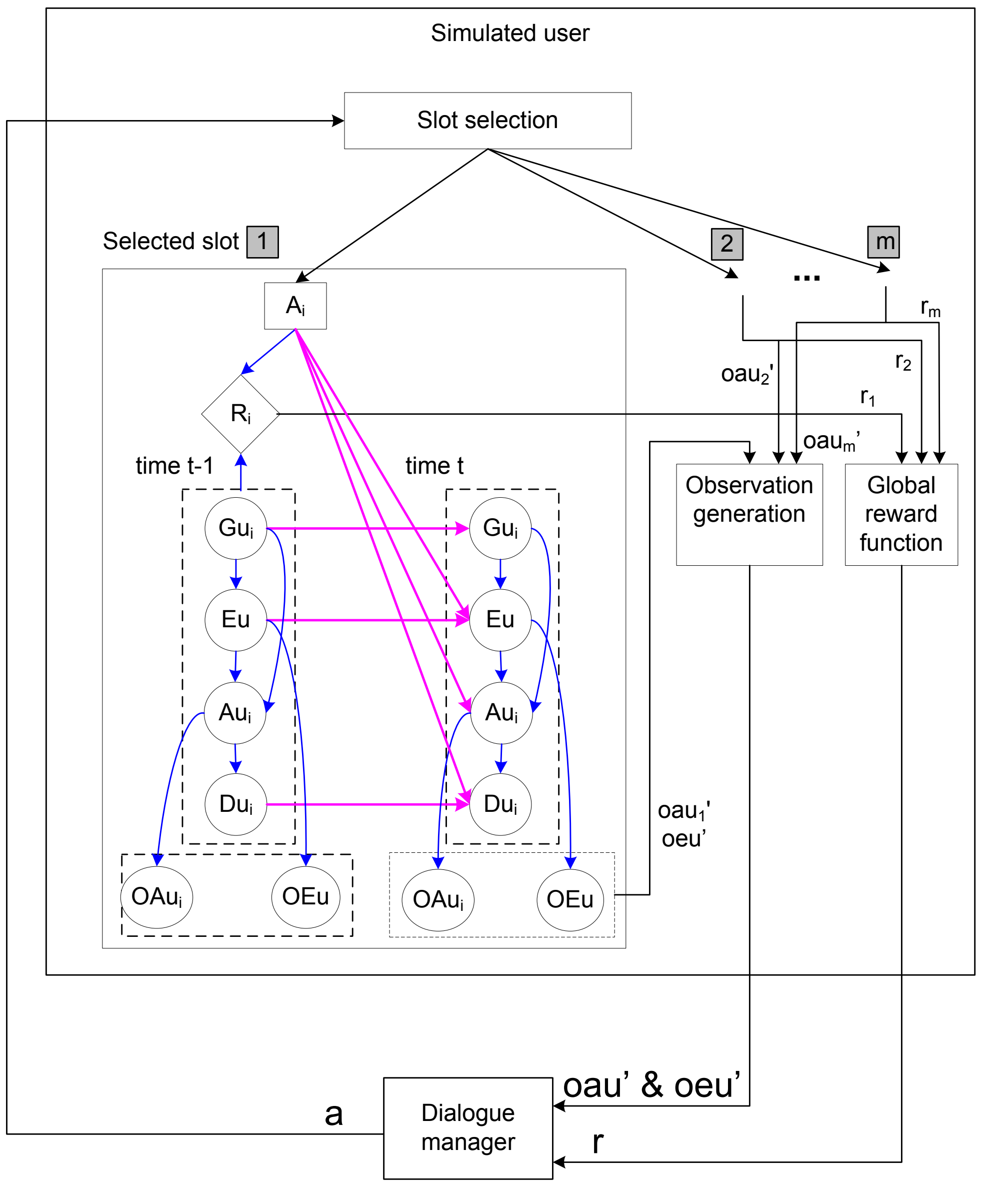


Figure 6.4: Simulated user model using DBNs. The user's state and action at each time-step are generated from the DBNs. Only the observation of the user's action and the user's affective state, and the reward are sent to the DM. The structures of slot DBNs are identical, therefore only one DBN is shown.

situation in the tunnel is very noisy". In this scenario, the user might be in a highly stressed situation, and can make errors in communicating with the system about the locations. Therefore, the DM needs to be robust to cope with errors from the user's mistakes and the system's input recognition and fusion modules [26, 66].

The route navigation example in Section 6.3 can be formulated as `n` slots $(f_1, ..., f_n)$ where $Gu_1 = Gu_2 = ... = Gu_n = V = \{v_1, v_2, ..., v_m\}$ (the set of all locations in and around the tunnel)[4]. The user's goal is represented as a function $\mathit{findRoute}(g_1, ..., g_n)$, with $g_i = x, i \in [1, n], x \in V$ and $g_i \neq g_j$ if $i \neq j$. In a frame-based DS, the user's goal states can be simplified as a set of n-tuples $\langle g_1, g_2, ..., g_n \rangle$. The user's affective states are five levels of the user's stressed situation: no stress (*no*), low stress (*low*), moderate stress (*moderate*), high stress (*high*), and extreme stress (*extreme*). We assume that the user only uses a limited set of main actions, that is to say that the set of user's speech acts is *answer*, *yes*, and *no*. Note that various combinations of the user's speech acts are allowed such as the user can respond *no* for some slots and respond *answer* for other some other slots in one turn. The user's grounding state is composed of two values *notstated* and *stated*. The set of system's speech acts is *ask*, *confirm*, *ok*, *fail*, *giveSolution*, and *stop*. The combination of these speech acts and the slots and slot values forms a complete system action such as $\mathit{ask}(f_1)$ or $\mathit{confirm}(f_2 = v_3)$. The two last special system acts (*giveSolution* and *stop*) are only used at the global DM level as being defined in Section 6.2.2. The user's objective is to find out the route description for `n` locations (the locations are known by the user). The system aims at showing the user the correct route as soon as possible.

6.3.1. Slot level dialogue manager representation

Using the formal formulation described in Section 5.4, the remaining features of slot f_i is represented by $Eu = \{e_1, e_2, ..., e_5\} = \{\mathit{no, low, moderate, high, extreme}\}$, $Au_i = \{\mathit{answer}(x),\ \mathit{yes}, \mathit{no} | x \in V\}$, $Du_i = \{\mathit{notstated, stated}\}$, $OAu_i = Au_i$, $OEu = Eu$, $A_i = \{\mathit{ask}, \mathit{confirm}(x),\ \mathit{ok}(x),\ \mathit{fail} | x \in V\}$. The flat POMDP model for this slot (including a special *end* state) is composed of $10m^2 + 20m + 1$ states, $2m + 2$ actions, and $5m + 10$ observations, where m is the number of locations in the tunnel.

The reward model for each slot is similar to the reward model of the single-slot route navigation example presented in Section 5.6 except that when slot `i` nominates an incorrect slot value the reward is `-50`. The high negative reward for selecting the incorrect slot value (`-50`) is used to motivate the dialogue manager agent to verify the information provided by the user when the user's stress level is high.

The probability distributions for each kDDNA are similar to that of the single-slot route navigation POMDP model described in Section 5.6 except that the user's stress model and user's action model are a bit different (because the stress variable E_u is now composed of five values). We also assume that the system action does not influence the user's stress. It means that the structure of kDDNAs is simpler. There are no links from the system action and the user's goal nodes to the user's stress node. This assumption does not circumvent the tractability of the kDDNA

[4]Our approach is generic and allows slots to have a different set of values

belief updating process as we will show in Section 6.4.1. Furthermore, we simplify the kDDNA network structure by pruning the observation nodes in the look-ahead slices[5].

Let $c(x)$ be the cardinal number of the element x in the list X (e.g., $c(x) = 1$ if x is the first element of list X), the simplified user's stress model (assuming that the system action does not influence the user's stress) is represented by (we consider set Eu is a list):

$$P(e'_u|e_u) = \begin{cases} 1 - p_{ec} & \text{if } e'_u = e_u, \\ p_{ec} & \text{if } |c(e'_u) - c(e_u)| = 1 \;\&\; e_u \in \{no, extreme\}, \\ \frac{p_{ec}}{2} & \text{if } |c(e'_u) - c(e_u)| = 1 \;\&\; e_u \in \{low, moderate, high\}, \\ 0 & \text{otherwise.} \end{cases} \tag{6.3}$$

An example of the user's stress model with $p_{ec} = 0.1$ is shown in Table 6.2. This model is consistent with the empirical work of stress recognition and modeling reported in *Liao et al.* [101]. The user's stress level changes gradually during the system-user interaction process. We simplified the influence of factors such as workload, trait, and context [101] to stress by parameter p_{ec}.

	e_u				
e'_u	no	low	moderate	high	extreme
no	0.9	0.05	0	0	0
low	0.1	0.90	0.05	0	0
moderate	0	0.05	0.90	0.05	0
high	0	0	0.05	0.90	0.1
extreme	0	0	0	0.05	0.9

Table 6.2: Handcrafted user's stress model with $p_{ec} = 0.1$, eu is the user's stress at time $t-1$ and eu' is the user's stress state at time t

[5]We thought that it was sufficient to maintain only the state variables for the look-ahead slices. However, this simplification does not improve the performance of the DDN-POMDP policy when we increase the number of look-ahead steps k. A further discussion about this issue is described in Section 6.4.4.

The user's action model is represented by:

$$P(a'_u|a, g'_u, e'_u) = \begin{cases} 1 & \text{if } a = ask,\ e'_u = no, \text{ and } a'_u = answer(g'_u), \\ 1 & \text{if } a = \mathit{confirm}(g'_u),\ e'_u = no, \text{ and } a'_u = yes, \\ 1 & \text{if } a = \mathit{confirm}(x),\ e'_u = no, \text{ and } a'_u = no, \\ 1 - p_1 & \text{if } a = ask,\ e'_u \in \{e_2, e_3, e_4\}, \text{ and } a'_u = answer(g'_u), \\ \frac{p_1}{|A_u|-1} & \text{if } a = ask,\ e'_u \in \{e_2, e_3, e_4\}, \text{ and } a'_u \neq answer(g'_u), \\ 1 - p_2 & \text{if } a = \mathit{confirm}(g'_u),\ e'_u \in \{e_2, e_3, e_4\}, \text{ and } a'_u = yes, \\ 1 - p_2 & \text{if } a = \mathit{confirm}(x),\ e'_u \in \{e_2, e_3, e_4\}, \text{ and } a'_u = no, \\ \frac{p_2}{|A_u|-1} & \text{if } a = \mathit{confirm}(g'_u),\ e'_u \in \{e_2, e_3, e_4\}, \text{ and } a'_u \neq yes, \\ \frac{p_2}{|A_u|-1} & \text{if } a = \mathit{confirm}(x),\ e'_u \in \{e_2, e_3, e_4\}, \text{ and } a'_u \neq no, \\ \frac{1}{|A_u|} & \text{if } (a = ok(y) \text{ or } \mathit{fail}) \text{ or } (e'_u = e_5 \ \&\ p_e > 0), \\ 0 & \text{otherwise,} \end{cases} \tag{6.4}$$

where $p_1 = p/K_{ask}$, $p_2 = p/K_{\mathit{confirm}}$, $p = [c(e'_u) - 2] \times \mathit{intensity} + p_e$, $\mathit{intensity} = \min\{0.1, (1 - 1/|A_u| - p_e) \div (|E_u| - 1)\}$, $x, y \in G_u$ and $x \neq g'_u$.

The main idea behind this handcrafted user's action model is as follows. When the user's stress level is *no*, no communicative errors are made. When the user's stress level is *extreme*, user's actions are selected randomly (except when $p_e = 0$, the stress does not have any influence on the user's action, i.e., the user's stress model for $e'_u \neq no$ is similar to that for $e'_u = no$). In the remaining cases, the user's action accuracy depends on the parameter p_e and the user's stress level. An example of the probability distributions when the system action at time $t-1$ is $a = ask$ and the user's goal at time t is $g'_u = v_1$ is shown in Table 6.3.

	e'_u				
a'_u	no	low	moderate	high	extreme
$answer(v_1)$	1	0.9	0.8	0.7	0.2
$answer(v_2)$	0	0.025	0.05	0.075	0.2
$answer(v_3)$	0	0.025	0.05	0.075	0.2
yes	0	0.025	0.05	0.075	0.2
no	0	0.025	0.05	0.075	0.2

Table 6.3: Extract of the user's action model ($a = ask$ and $g'_u = v_1$) with $m = 3, p_e = 0.1$, and $K_{ask} = 1$.

6.3.2. Global dialogue manager representation

Let $F = \{f_1, f_2, ..., f_n\}$ be the set of slots, the system action set and the observation set of the user actions are now represented by:

- $A = \{ask(I), \mathit{confirm}(J),\ \mathit{giveSolution}(L),\ \mathit{stop}\}$

- $OAu = \{answer(J), yes, no\}$

where $I \subseteq F$, $J \subseteq L$, and $L = \{(f = x)|f \in F, x \in V\}$. The AS generates the global system action based on the following rules (applying the first rule that satisfies the set of nominated actions):

1. If all slots nominate *ask* action then the global action is *askopen*;
2. If all slots nominate *confirm* action then the global action is $\textit{confirmall}(f_1 = x, f_2 = y, ..., f_n = z)$, where $x, y, z \in V$;
3. If all slots nominate *ok* action then the global action is $\textit{giveSolution}(f_1 = x, f_2 = y, ..., f_n = z)$, where $x, y, z \in V$;
4. If some slots $(f_i, f_j, ..., f_k)$ nominate *confirm* action with the values $(x, y, ..., z)$ then the global action is $\textit{confirm}(f_i = x, f_j = y, ..., f_k = z)$, where $x, y, z \in V$;
5. If some slots $(f_i, f_j, ..., f_k)$ nominate *ask* action then the global action is $ask(f_i)$;
6. Otherwise, the global action is *stop*.

We can also divide the *confirm* action into explicit confirm (denoted by *econfirm*) and implicit confirm and ask (denoted by *iconfirm_ask*). When some slots nominate *confirm* action and some other slots nominate *ask* action, the action selector can either select *econfirm* or *iconfirm_ask* action. Further discussion about this issue is beyond the scope of this chapter.

Our prototype DM is a distributed multi-agent system developed using the Java programming language and the middleware iROS platform[6]. The DM agent exchanges messages with other input and output agents using the iROS Event Heap, a blackboard-like communication mechanism. The kDDNAs are created and integrated with the DM using the SMILE library[7]. A dialogue example of the 10-slot case $(n = 10, m = 10, p_{gc} = 0, p_{ec} = p_e = p_{oa} = p_{oe} = 0.1, K_{ask} = 1, K_{confirm} = 10, k = 1)$ is described in Appendix B.2.

6.4. Evaluation

In this section we evaluate our DDN-POMDP approach on several important issues for building a realistic DS. Section 6.4.1 is about the tractability of the belief monitoring and action selection processes when we scale up number of slots and slot values. The benchmark of the DDN-POMDP, approximate POMDP, and handcrafted policies for a 1-slot case is described in Section 6.4.2. The evaluation of the DDN-POMDP approach for a multi-slot case is presented in Section 6.4.3. Section 6.4.4 examines the performance of the DDN-POMDP policy with different look-ahead steps.

[6]http://sourceforge.net/projects/iros[accessed 2008-03-28]
[7]http://genie.sis.pitt.edu[accessed 2008-03-28]

6.4.1. Tractability

Three computational tractability issues are tackled in this section: (i) scaling up the number of slots; (ii) scaling up the number of slot values; and (iii) comparison of the belief updating time with different kDDNA structures.

We conducted various experiments with different numbers of slots. In a 1,000 slots test case, our DDN-POMDP DM is able to handle the information provided by the user in real-time. This is because the number of slots does not influence the complexity of the kDDNAs. To our knowledge, the ability to handle such a large number of slots is sufficient for developing all practical frame-based dialogue problems.

A detailed description of the 1000-slot test case (each slot has 10 values) is as follows. First, one thousand POMDP parameter files are generated. Each file contains a compact representation of a slot POMDP problem $(S, A, Z, T, O, R, \gamma)$. Although the POMDP parameter files are identical, we treat them as if they are different to guarantee that the test is applicable to frame-based dialogue problems of which slots' structures might be different. Second, when the system starts, the DM reads these files and creates kDDNAs. Each slot has 11 kDDNAs (only kDDNAs for *ask* and *confirm* actions are sufficient because when the system selects *ok* or *fail* action, no further belief updates are required). The total number of the created kDDNAs is 11000. The time taken to create, initialize probability distributions, and store these 11000 kDDNAs (using the SMILE library) in the computer memory is only 14 seconds[8]. Third, the initial belief states of slots are taken from the POMDP parameter files and updated into the DIS. Fourth, all slots nominate greedy action (*ask* action) and the *askopen* action is sent to the output module. Note that we have not yet executed any kDDNA belief update at this moment. Finally, when the DM receives the observation of the user's action and emotion, only relevant kDDNAs are activated for the belief update and on-line action selection tasks. With $k = 0$, the time taken to update 200 different kDDNAs simultaneously is around one second.

Scaling up the number of slot values increases the complexity of the kDDNAs. Using the SMILE library, our DDN-POMDP dialogue manager ($k = 0$) can handle problems of up to 500 slot values in real-time (belief update time for one slot is smaller than one second, Fig. 6.5). Note that the kDDNA belief update is a combined process: computing the belief state[9] and selecting the best slot action. Figure 6.5 also shows that the kDDNA belief update time is longer when the number of slots being processed simultaneously increases. A solution for many slots updating simultaneous is to deploy more computers and to divide the belief update task over these computers. This solution is not expensive, because in reality, the information provided from the user is usually relevant to a small number of slots. Because we make an independence assumption between slots, when the system gains information of a slot, the probability distributions of the values of slots relevant to it are not updated. This can lead to the case that two slots nominate the same value and the action selector must decide which one will be included in the global system action.

[8]The test was conducted on a PC with 3.2 GHz CPU and 2 GB of RAM. Hereafter, this PC configuration is used as the default platform for our experiments. Otherwise, we will state explicitly.

[9]Similar to the belief monitoring in a standard POMDP model.

Unfortunately, the kDNNA belief updating time increases exponentially with the number of look-ahead steps (Fig. 6.6). This suggests that when $k \geq 2$, it is better to conduct the policy search off-line and integrate the result (either in the form of a policy graph or an alpha file) into the dialogue management module for action selection. Further details about the off-line policy search issue will be explored in the future.

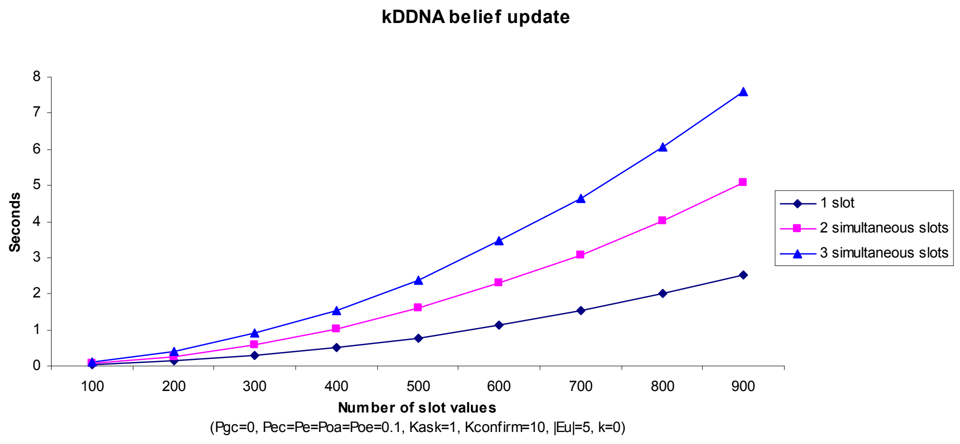


Figure 6.5: Average kDDNA belief updating time over 10 runs in seconds with different numbers of slot values using the SMILE library. The best result is found using the Pearl exact inference algorithm and the Find Best Policy update.

Table 6.4 shows the belief updating time over 10 runs on different kDDNA's structure. The kDDNA1's structure is the simplified structure used for our next experiments (without linkage from the user's action and user's goal nodes to the user's emotion node). The kDDNA2's structure is described in Figure 6.2b. The belief updating time of the kDDNA2 is a bit slower when $k = 1$.

When a slot has thousands of values (called many-value slot), one solution is to bundle slot values into partitions and to update the belief directly on these partitions as proposed by *Young et al.* [185]. In Chapter 3, we formulated the many-value slot as a list processing slot. This is based on our practical experience of building real-world frame-based dialogue systems. For some slots, we cannot determine all possible values at the design time such as the type of foods in the restaurant search application or the title of movies to be recorded in a smart-home application. Therefore, values of these slots are defined as ordinal numbers such as *one*, *two*, ..., *ten*, *previous*, and *next*. The DM and the user then only work with these list processing values. A mapping between these values and the real values is done automatically by the DM. However, whether this idea is useful for statistical dialogue management models, where slot values are represented as a probability distribution, is still an open issue. We defer the study of this representation to the future.

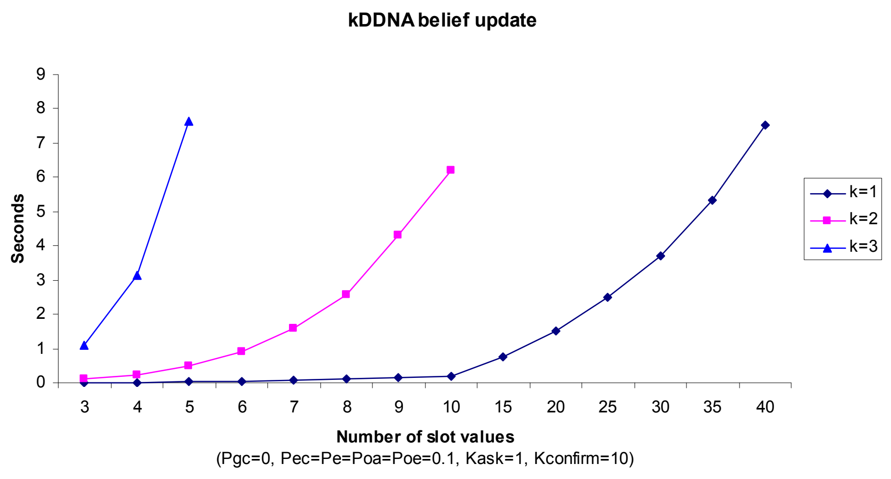


Figure 6.6: Average kDDNA belief updating time over 10 runs with different look-ahead steps.

k=0			k=1		
m	kDDNA1	kDDNA2	m	kDDNA1	kDDNA2
100	0.0313	0.0312	3	0.0485	0.0500
200	0.1281	0.1282	5	0.1063	0.1109
300	0.2875	0.2860	10	0.2031	0.3922
400	0.5063	0.5046	15	0.7438	0.8985
500	0.7859	0.7860	20	1.5375	1.5922
600	1.1407	1.1390	25	2.5968	2.6000
700	1.5328	1.5373	30	3.8937	3.9124
800	1.9906	1.9906	35	5.5624	5.9797
900	2.5297	2.5375	40	7.8593	8.2421

Table 6.4: Average belief updating time in seconds over 10 runs with different kDDNA structures, with m slot values. The structure of the *kDDNA1* is the simplified structure without linkage from the system action and user's goal nodes to the user's emotion node. The structure of the *kDDNA2* is the complete structure as described in Figure 6.2.

6.4.2. Comparison with approximate POMDP and handcrafted policies (single slot)

The performance of the DDN-POMDP dialogue policy depends on both the global DM and the slot level DM (Section 6.2). In this section, we evaluate the performance of the slot level DM by comparing the DDN-POMDP dialogue policy with an approximate POMDP policy and three handcrafted dialogue strategies HC1, HC2, and HC3 (Fig. 6.7) for the 1-slot case. HC1 is also the optimal dialogue policy when $p_{gc} = p_e = p_{oa} = 0$ (the user's goal does not change; stress has no influence on the user's action and no error in observing the user's action i.e., the speech recognition and spoken language understanding errors are equal to 0).

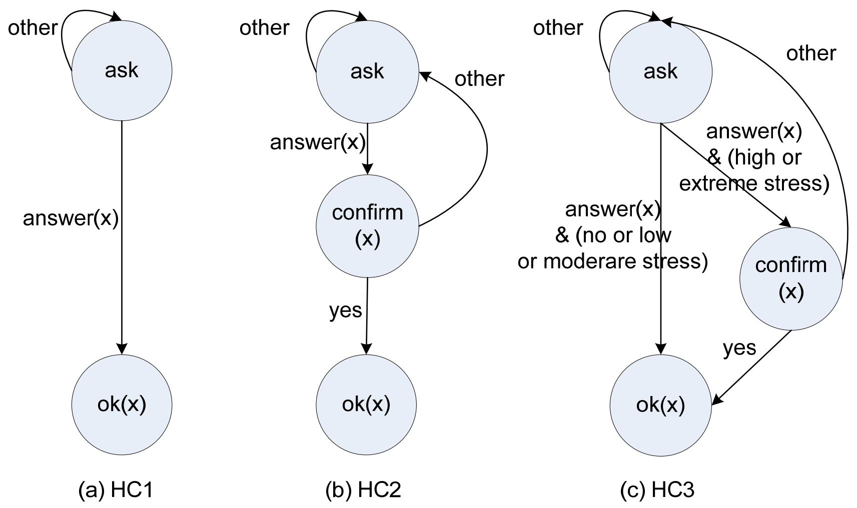


Figure 6.7: Three handcrafted dialogue strategies for the 1-slot case (x is the slot value): (a) first *ask* and then select *ok* action if *userSpeechAct* = *answer* (otherwise *ask*), (b) first *ask*, then *confirm* if *userSpeechAct* = *answer* (otherwise *ask*) and then select *ok* action if *userSpeechAct* = *yes* (otherwise *ask*), (c) first *ask*, then *confirm* if *userSpeechAct* = *answer* & *oeu* = *high* or *extreme* and select *ok* action if *userSpeechAct* = *yes*.

The evaluation is conducted by letting each dialogue policy interact with the same simulated user (the simulated user model is constructed using the DBN described in Fig. 6.4). We conducted a large number of dialogue episodes (10,000) to guarantee the statistical significance. Error bars in the figures (from 6.8 to 6.11) show the 95% confidence level.

Figure 6.8 shows the average return of the DDN-POMDP policies and a best approximate POMDP policy computed using Perseus for a simple dialogue problem (one slot, three values). The probability distributions and the internal reward models[10] of the DDN-POMDP and DDN-POMDP* models are similar to the one used in the simulated user except that in the DDN-POMDP* model, we set a lower internal reward when the DM gives an incorrect solution ($rOkIncorrect = -200$). However, Figure 6.8 also shows that the performance of the DDN-POMDP policy is significantly improved (close to the standard POMDP policy) when we tune its internal reward model. The idea to tune the DDN-POMDP internal reward model is based on our practical experience. The behavior of the DDN-POMDP policy is sensitive to its internal reward model. When the $rOKIncorrect$ value is low, the DDN-POMDP DM is more sensitive in confirming the information provided by the user before giving a solution. It is clear that, when the number of slots and slot values is small, the POMDP dialogue policy is the best solution, because the policy computed by the approximate POMDP solver is similar to the DDN-POMDP policy with $k = \infty$. Note that when computing the average return of all three cases, we do not change the reward model of the simulated user which is defined in Section 6.3.1. The average return of the DDN-POMDP is low when $p_e = 0.1$ or 0.2. We looked at the behavior of this policy in case $p_e = 0.1$ and found that it uses the *confirm* action when the observation of the user's affective state is *high* or *extreme* (similar to the HC3 policy). On the other hand, both DDN-POMDP* and POMDP policies use the *confirm* for all observations of the user's affective state (similar to the HC2 policy). It means that in this case, it is better to *confirm* after the user provides a location no matter what the observation of the (simulated) user's affective state is.

We conducted further experiments with different internal reward values of providing an incorrect solution ($rOkIncorrect \in [-10^7, -10]$, Fig. 6.9). At the beginning, the average return increases monotonously with the absolute value of *rOkIncorrect*. The DDN-POMDP policy performs best when $rOkIncorrect \in [-1000, -200]$. When $rOkIncorrect < -1000$, the average return degrades. Interestingly, there is a correlation between the high negative value of *rOkIncorrect* and the number of times the system selects *confirm* action before giving a solution to the user.

Figure 6.10 shows the average return of five dialogue strategies when the probability of the user's action error being induced by stress p_e changes from `0` (stress has no influence on the user action selection) to `0.8` (stress has high influence on the user action selection). The results of the average return (Fig. 6.10) show that with a 95% confidence level the DDN-POMDP dialogue policy outperforms all other remaining dialogue strategies when $p_e \geq 0.1$[11]. The DDN-POMDP copes well when the user's action error being induced by stress increases. An example of the interaction between the DDN-POMDP DM and the simulated user (three dialogue episodes) is shown in Appendix B.1.

Figure 6.11 shows that the DDN-POMDP dialogue policy also copes well with

[10]Note that we distinguish two types of reward models. The first type is the reward model from the POMDP environment. It is called the *external reward model* or simply *reward model*. This type of reward model is described in Section 6.3.1. The other type of reward model is placed inside the agent or the DM. It is called *internal reward model* [163].

[11]Perseus fails to find a near-optimal policy because of the run out of memory problem (tested on a PC with 3.2 GHz CPU and 2 GB of RAM)

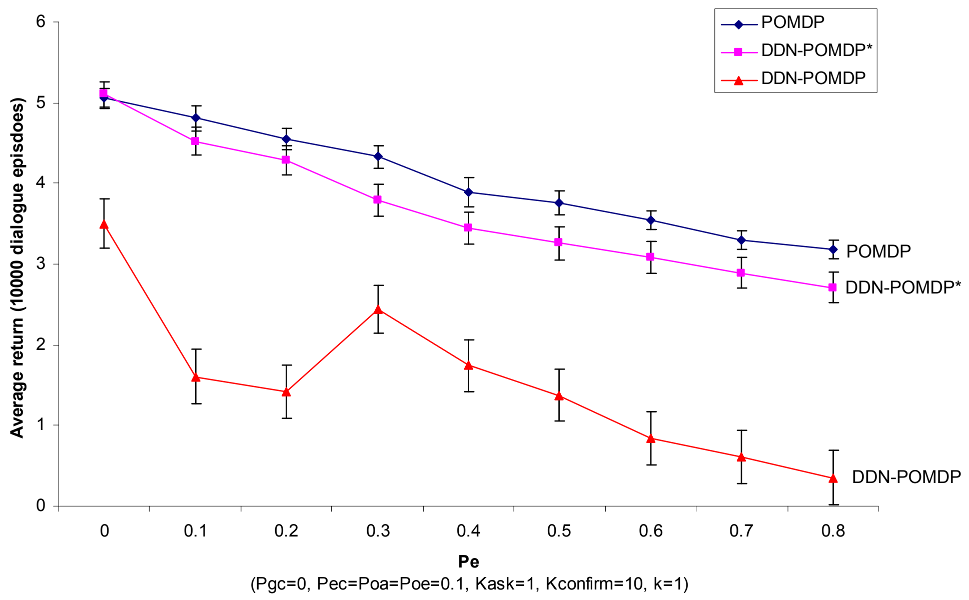


Figure 6.8: Average return of the DDN-POMDP policies and the approximate POMDP policy for a one slot, three values case. Error bars show the 95% confidence level. The caption "*" at the end of a policy title indicates that the internal reward function of the dialogue manager associated with the policy is tuned.

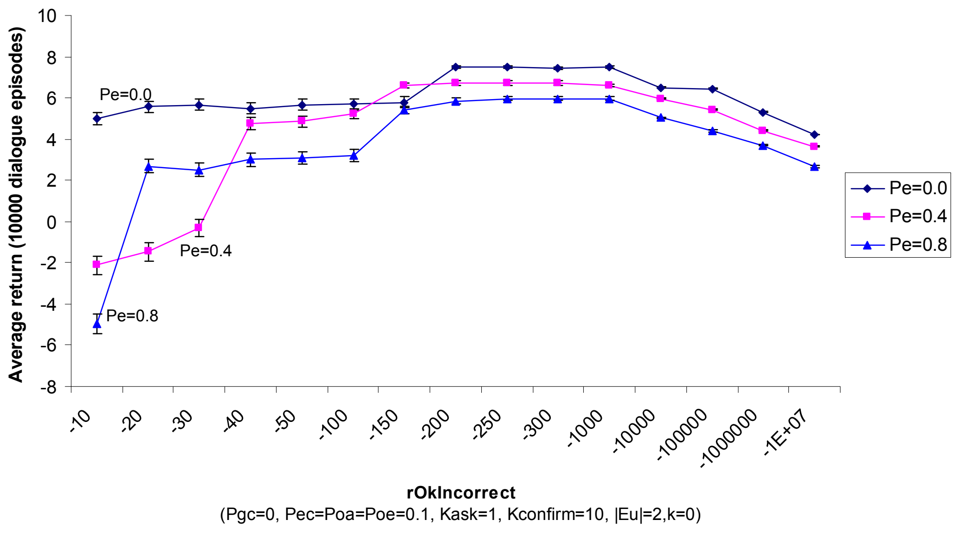


Figure 6.9: Internal reward optimization for a one slot, three values case. Experiments were conducted with p_e in the range from 0 to 0.8. All average returns are optimal in the range $[-1000, -200]$. Only three lines are shown for the purpose of clarity. Error bars show the 95% confidence level.

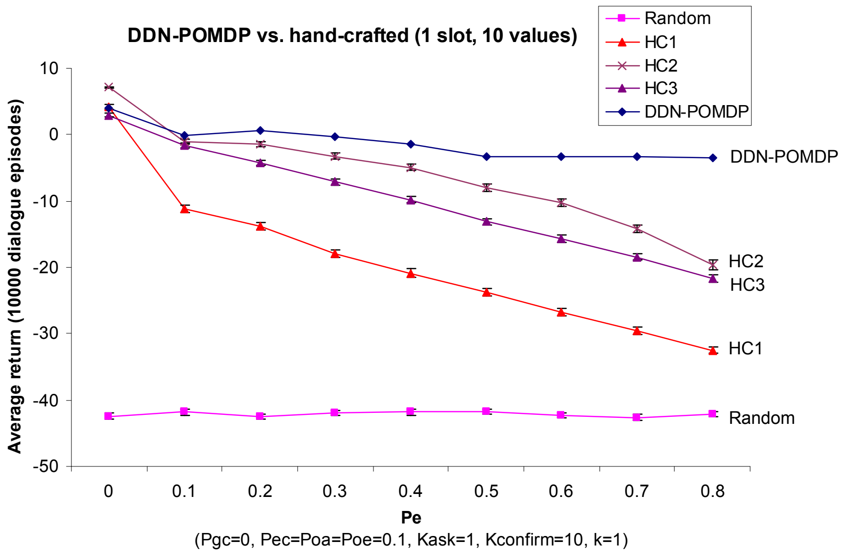


Figure 6.10: Average return vs. the user's action error being induced by stress (p_e). Error bars show the 95% confidence level.

the observation error of the user's action p_{oa}. When the observation error p_{oa} is too high ($p_{oa} \geq \texttt{0.6}$), the DDN-POMDP DM always selects the *fail* action therefore the average return is a constant (equal to `-4`). The *fail* action in this example means that the system stops the dialogue and transfers the user's request to a human operator. Note that whether the system selects the *fail* action or not depends on the negative reward associated with this action. In the above example, selecting the *fail* action at the beginning is equivalent to getting the correct location after 13 turns (in both cases, the system receives the same amount of reward). One interesting point is that although dialogue policy HC2 copes well with the change of p_e (Fig. 6.10), its performance decreases rapidly when p_{oa} increases (Fig. 6.11).

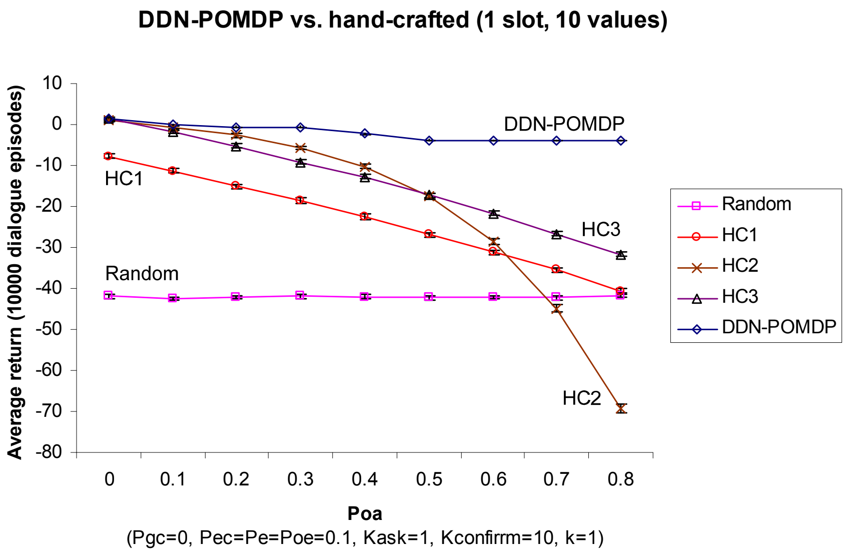


Figure 6.11: Average return vs. the observation error of the user's action p_{oa}. Error bars show the 95% confidence level.

We also evaluated the performance of the DDN-POMDP dialogue policy with a number of different setups: (1) the transition and observation probability distributions of the DDN-POMDP and the simulated user are the identical (source model); and (2) the probability distributions of the DDN-POMDP are fixed with p_e = `0.15`, `0.45`, and `0.6` respectively. Figure 6.12 shows that the DDN-POMDP dialogue policy can cope robustly with the variant of the user's model (the POMDP policy is shown as an upper-bound of the performance). Note that the performance of the DDN-POMDP policy in this figure is different when compared with the one in Figure 6.10 because we use a different (external) reward model[12]. Reward for selecting the incorrect slot

[12]This does not influence the robustness of the DDN-POMDP policy when tested with the reward model presented in Section 6.2.1.

value is `-10`, and `-5` for action *fail*. The user's affective states are classified simply as *nostress* and *stress*. The reward function used in Figures 6.10, 6.11, and 6.12 is not tuned except that we set the internal reward for *confirm* action on the user's goal to -0.99 (instead of -1). The modified internal reward function helps the DDN-POMDP DM avoid a pitfall with the *confirm* action. We detected that when we incorporated the external reward model into the DDN-POMDP DM, the system might repeatedly uses the *confirm*(v_1) action. The DDN-POMDP policy after including this modified internal reward function acts on goals.

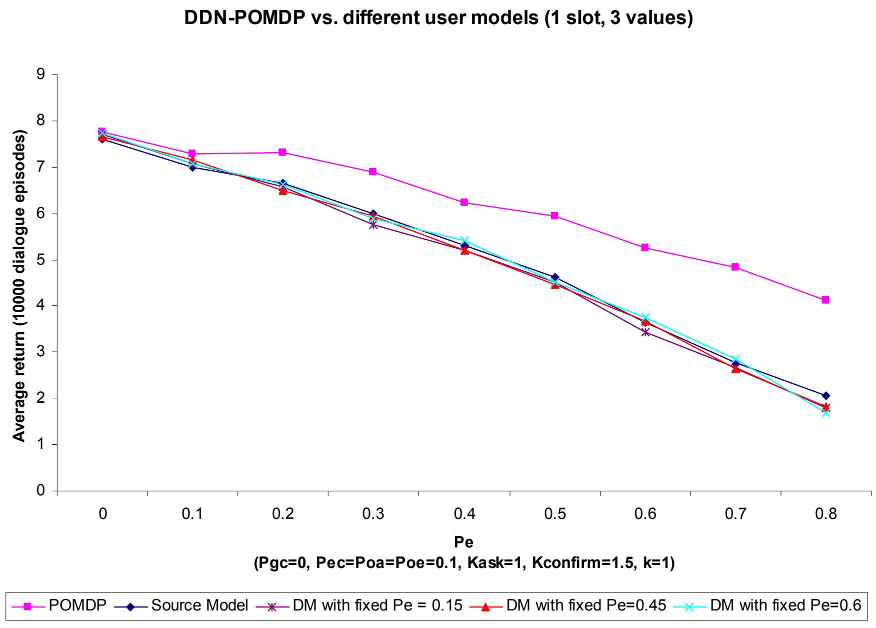


Figure 6.12: The performance of the DDN-POMDP policy with fixed p_e

6.4.3. Comparison with enhanced handcrafted policies (two slots)

This section evaluates the performance of the DDN-POMDP versus enhanced handcrafted policies for a 2-slot case. The user's model at the global level is adapted from the SACTI training corpus [177]. The model is shown in Table 6.5.

Note that we extend several user's options from the original model to illustrate the ability of our method to cope with different variabilities of the user's response. For example, when the user is asked on a slot, he might answer on another slot. Similarly, when the system confirms a slot (*Do you mean the second location is A?*), the user might respond to both slots (*No, A is the first location.*).

System action	User action	Probability
ask open	no response	0.013
	respond to the first slot	0.207
	respond to the second slot	0.207
	respond to both slots	0.573
ask slot f	no response	0.013
	respond to slot f	0.843
	respond to the remaining slot	0.044
	respond to both slots	0.100
confirm slot f	no response	0.013
	respond to slot f	0.943
	respond to both slots	0.044
confirm all	no response	0.013
	respond to both slots	0.987

Table 6.5: User's model for slot selection adapted from the training model of the SACTI corpus [177] with several extensions

Based on the three 1-slot handcrafted dialogue strategies presented in Figure 6.7 we implemented three multi-slot handcrafted dialogue strategies HC1+ (Alg. 5) , HC2+ (Alg. 6), and HC3+ (Alg. 7). All three strategies start with the *askopen* action. Every time the dialogue manager receives an observation it filters the incoherent responses before the action selection process to minimize the unnecessary dialogue turns. For example, if two slots have the same value, only the slot relevant to the last system action is selected. Further, based on the experience with the 1-slot case, we improve the performance of all three handcrafted strategies by selecting *stop* action when the observation of the user's emotion is *extreme* and $p_e > 0$, similar to the behavior of the DDN-POMDP policy. In addition, the inter-turn inconsistency is also handled to prevent the system from providing incoherent solutions such as $giveSolution(f_1 = x, f_2 = x)$. Note that the *updateUserState* function of the HC3+ algorithm is similar to the one of the HC2+ algorithm except the *updateUserState* function of the HC3+ changes the dialogue state *du* from -1 to 1 when the observation of the user's action is *answer* and the observation of the user's emotion is *no* or *low* or *moderate* stress.

Figure 6.13 shows the average return of the DDN-POMDP* (the best internal reward *rOkIncorrect* for $k = 0$ is -250 and for $k = 1$ is -500)[13] and handcrafted strategies. It is still valid that when the error rate is high, the DDN-POMDP* strategy performs better than the enhanced version of the handcrafted counterparts. Because the behavior of the handcrafted strategies does not depend on the numeric value of the reward function, we further compare the average number of turns and the percentage of correct goals. Table 6.6 shows that the percentage of achieved goals of the policy HC2+ is high as expected. The DDN-POMDP* policy gained a higher number of achieved goals with fewer number of turns compared to the HC2+ policy.

[13]We also re-tuned the internal reward model at each value p_e. The re-tuned policy performs slightly better when compared with the best policy with a fixed internal reward *rOkIncorrect*.

Algorithm 5: HC1+(oau, oeu) (Enhanced handcrafted strategy 1)

```
Input: observation of the user slot actions $oau$, observation of the user's
       emotion $oeu$
Output: system action $a$
begin
    initializeEpisode();
    $oau$ := filterInconsistency($oau$);
    updateUserState($a$, $oau$);
    if $du = [0\ 0]$ then $a$ := giveSolution($gu$);
    else $a$ := ask first open slot;
    return a;
end
initializeEpisode()
begin
    User slot goals $gu := [-1 - 1]$;
    // -1:  not provided or open, otherwise:  index of slot goal
    Slot dialogue states $du := [-1 - 1]$;
    // -1:  notstated or open, 0:  stated, 1:  confirmed
    System action $a$ := askopen;
end
```

Algorithm 6: HC2+(oau, oeu) (Enhanced handcrafted strategy 2)

```
Input: observation of the user slot actions $oau$, observation of the user's
       emotion $oeu$
Output: system action $a$
begin
    initializeEpisode();
    $oau$ := filterInconsistency($oau$);
    updateUserState($a$, $oau$);
    if $du = [1\ 1]$ then $a$ := giveSolution($gu$);
    else if some slots are stated then $a$ := econfirm(statedSlots);
    else $a$ := ask first open slot;
    return a;
end
```

Algorithm 7: HC3+(oau, oeu) (Enhanced handcrafted strategy 3)

```
Input: observation of the user slot actions $oau$, observation of the user's
       emotion $oeu$
Output: system action $a$
begin
    initializeEpisode();
    $oau$ := filterInconsistency($oau$);
    updateUserState($a$, $oau$, $oeu$);
    if $du = [1\ 1]$ then $a := giveSolution(gu)$;
    else if some slots are stated then $a := econfirm(statedSlots)$;
    else $a$ := ask first open slot;
    return a;
end
```

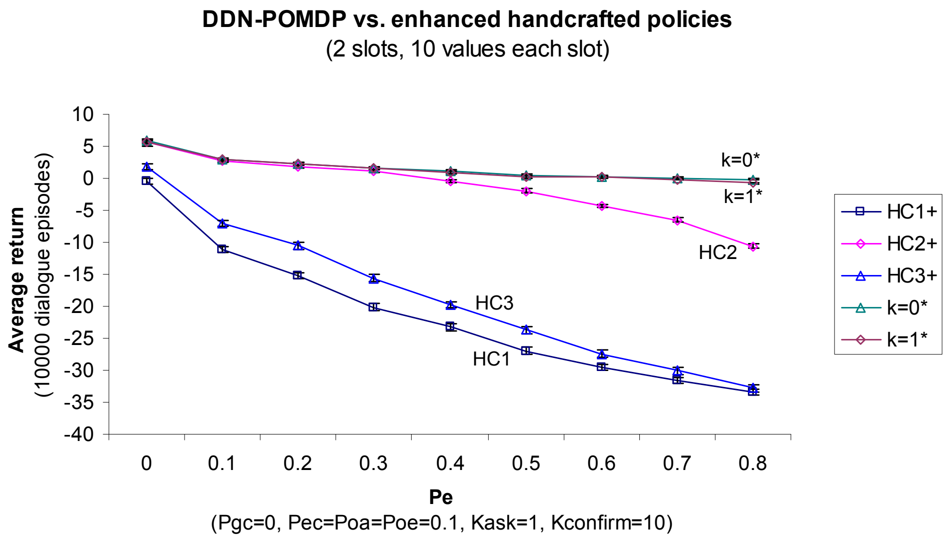


Figure 6.13: Average return vs. the user's action error being induced by stress (p_e) for a 2-slots case. Error bars show the 95% confidence level.

	% achieved goals					Average number of turns per episode				
p_e	HC1+	HC2+	HC3+	k=0*	k=1*	HC1+	HC2+	HC3+	k=0*	k=1*
0.0	0.850	0.986	0.900	0.998	**0.998**	2.507	4.574	3.192	4.861	5.277
0.1	0.533	0.802	0.611	0.807	**0.809**	2.479	4.873	3.065	5.164	5.337
0.2	0.473	0.795	0.558	**0.814**	0.807	2.516	5.387	3.186	5.763	5.778
0.3	0.392	0.789	0.481	**0.805**	0.802	2.548	5.885	3.317	6.183	6.265
0.4	0.341	0.780	0.415	**0.800**	0.798	2.587	6.766	3.486	6.411	6.692
0.5	0.278	0.772	0.349	**0.790**	0.789	2.611	7.730	3.656	6.936	7.251
0.6	0.236	0.742	0.293	0.792	**0.806**	2.625	8.991	3.806	7.241	7.744
0.7	0.198	0.740	0.242	0.796	**0.802**	2.637	10.614	3.912	7.757	8.304
0.8	0.179	0.701	0.206	0.796	**0.801**	2.659	12.963	4.130	8.202	8.874

Table 6.6: Percentage of achieved goals and average number of turns per episode. Bold numbers show the highest percentage of achieved goals for each p_e

6.4.4. Look-ahead performance

Figure 6.6 shows that the belief update time increases in the dimension of the number of look-ahead slices k. A follow-up question is how does k influence the performance of the DDN-POMDP policy. The result conducted in Section 6.4.3 does not show a clear advantage of the DDN-POMDP policy conducted with $k = 1$ compared with the one conducted with $k = 0$. This is because we do not incorporate the observation nodes into the look-ahead slices of the kDDNA network structure (as mentioned in Section 6.3.1). This section explores the performance of DDN-POMDP policy on the full kDDNA network structure (Fig. 6.2b).

Figure 6.14 shows that the performance of the DDN-POMDP policies increases in the value of k as expected. The DDN-POMDP policies with $k = 1$ and $k = 2$ outperform the one with $k = 0$ when $p_e \geq 0.1$. The DDN-POMDP policy with $k = 2$ outperforms the one with $k = 1$ when $0.1 \leq p_e \leq 0.3$.

Looking at the logfile of the dialogue simulation (Fig. 6.15) we found that when $p_e \geq 0.1$, the DDN-POMDP-2 policy (hereafter the DDN-POMDP with k look-ahead step is denoted by DDN-POMDP-k) starts to confirm the location provided by the user while the DDN-POMDP-1 and DDN-POMDP-0 policies only change to this behavior when $p_e \geq 0.4$ and $p_e = 0.8$ respectively.

We further examined the performance of the DDN-POMDP policies versus the observation error p_{oa}(Fig. 6.16). The DDN-POMDP-2 policy outperforms the other cases (i.e. $k = 0$ and $k = 1$) when $0.1 \leq p_{oa} \leq 0.4$. However, this trend does not hold when $p_{oa} \geq 0.5$. It suggests that when the observation error is too high, looking ahead two steps (i.e., $k = 2$) is not sufficient to derive a distinctively good system action.

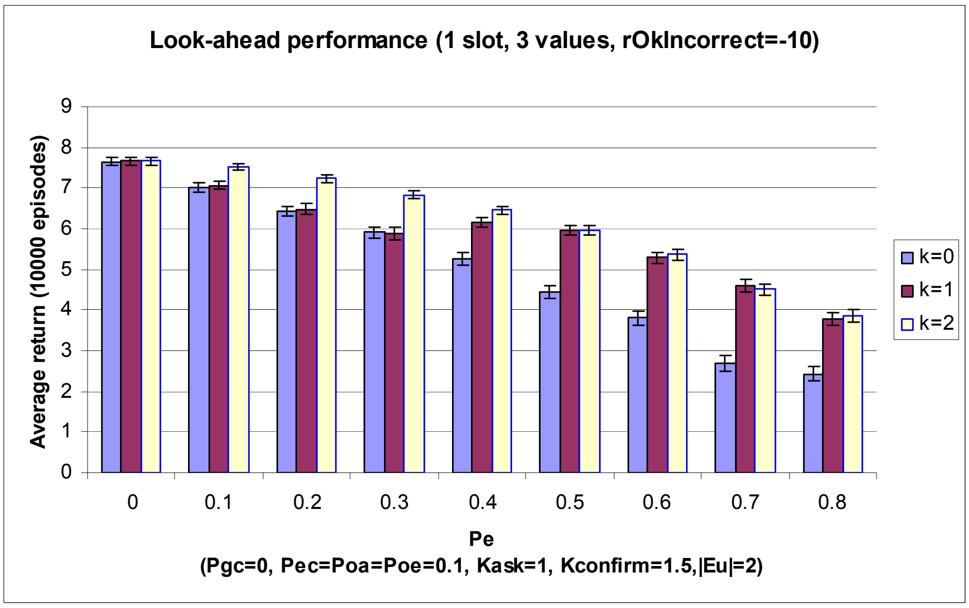


Figure 6.14: Performance of the DDN-POMDP policy with different look-ahead values k

6.5. Discussion

When the application domain has a small number of slots and slot values, a POMDP DM is an ideal solution as described Chapter 5. In this case, the POMDP dialogue policy outperforms the proposed DDN-POMDP dialogue policy (Fig. 6.8). However, when the application is more complex, we cannot compute a near-optimal POMDP dialogue policy using standard approximate POMDP algorithms such as Perseus. Thus, our solution is a good starting point to allow for creating a workable prototype DM. The DDN-POMDP method can also be applicable for spoken dialogue systems (by removing the Eu and OEu nodes from the state and observation sets). Therefore, it would be interesting to compare our method with the CSPBVI method [176].

Furthermore, the ability to handle many slot values also helps when modeling complex affective states of the user. Suppose that the user's emotion model is represented by a set of six basic emotions (*happiness, surprise, anger, sadness, fear*, and *disgust*) and ten levels of intensity for each emotion $(0.0, 0.1, ..., 0.9)$. If we integrate this emotion model into the state space, the number of states is bigger than 10^6 (equivalent to a kDDNA model in Figure 6.2 with 300 slot values), which cannot be solved by current approximate POMDP algorithms such as Perseus. This problem can easily be solved by our DDN-POMDP approach.

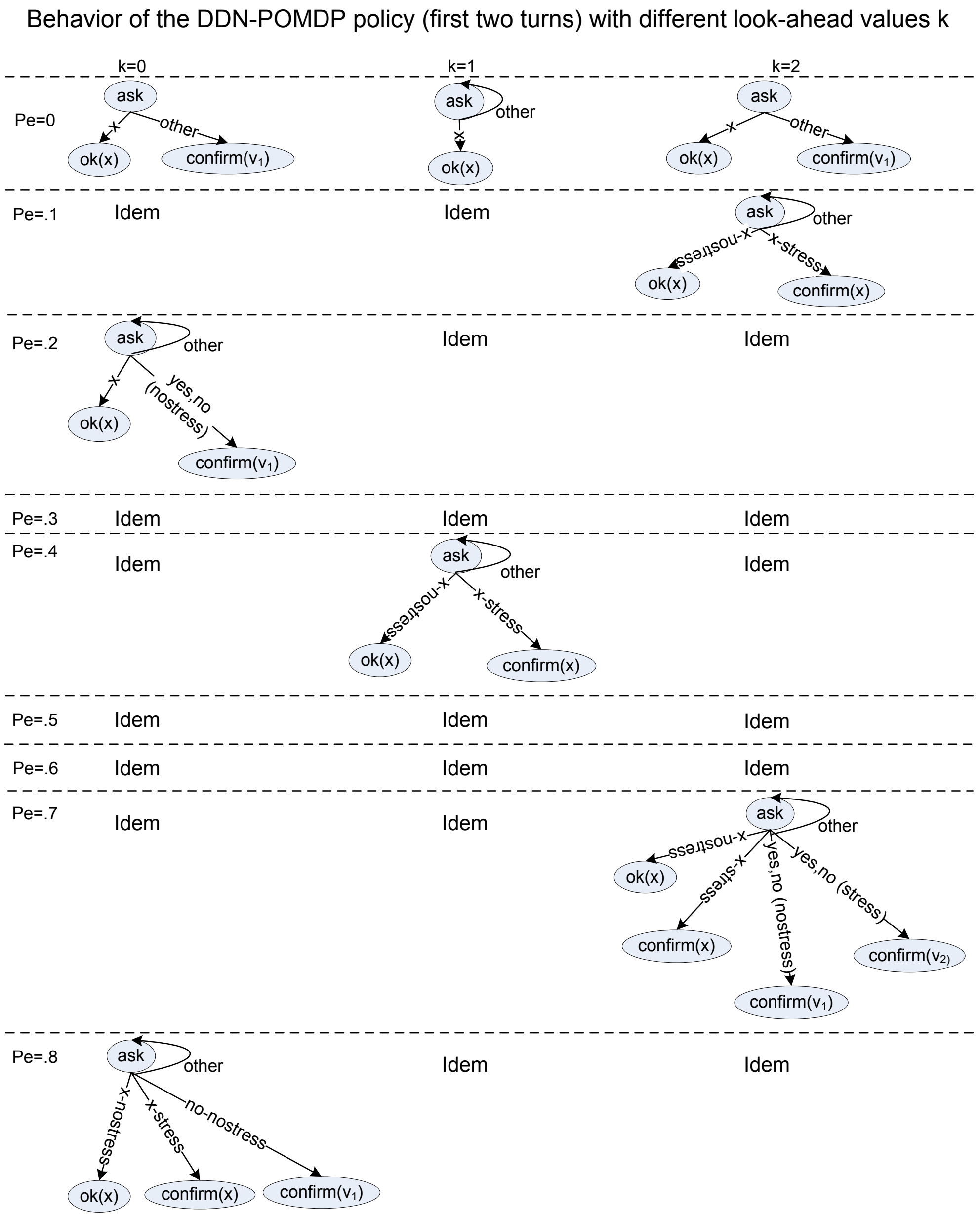


Figure 6.15: Behavior of the DDN-POMDP policy (first two turns) with different look-ahead steps

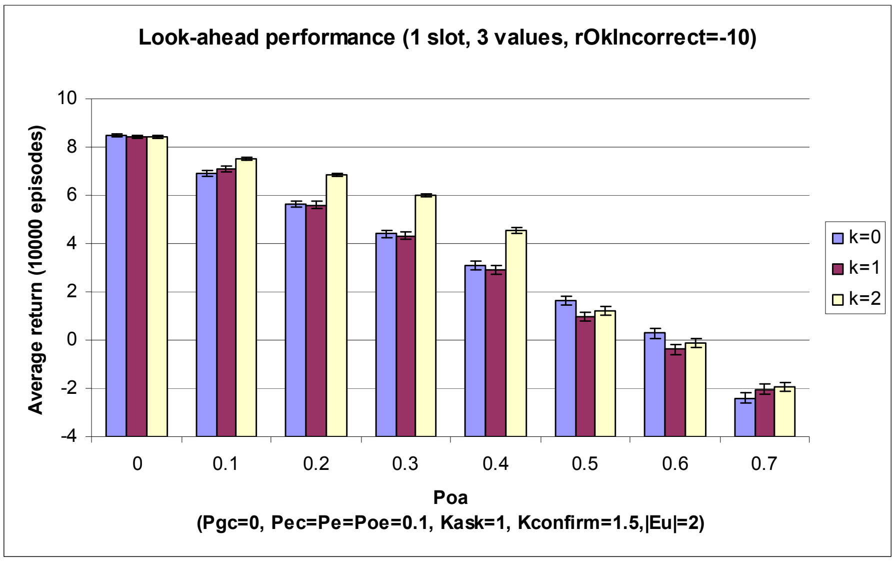


Figure 6.16: Performance of the DDN-POMDP policy with different look-ahead values k

6.6. Conclusions

In this chapter, we have presented an approach to developing a tractable affective dialogue model for probabilistic frame-based DSs. Our approach is based on the POMDP and DDN techniques. The DM is divided into two parts: the slot level DM and the global DM. The first part is composed of `n` slots where each slot is modeled as a factored POMDP and approximated as a set of DDNs. The second part (i.e. the global DM) is composed of two components: (1) the DIS, the active memory of the DM, is responsible for updating and memorizing the current probability distributions of the user's goal, the user's affective state, the user's action, the user's grounding state of all slots, and the recent observation of the user's action and affective state; (2) the AS is responsible for aggregating the system's slot actions nominated by the slots. The DM activity process is explained by a cycle of four steps (Fig. 6.3).

Two key features of our model are *realistic affective modeling* and *ability to handle many slots and slot values*. Our dialogue model takes into account the user's affective state with *uncertainty* and *dynamic aspects* (i.e., change over time). In previous work the user's affective state was either considered as fully observable [8, 80, 129] or static (i.e., does not change during the system-user interaction [130]). The ability to handle many slots and slot values helps when modeling real-world frame-based problems. Our current implemented prototype DM can handle problems with the size of hundreds of slots, where each slot might have hundreds of values. Possible solutions for very large numbers of slot values were also suggested.

We conducted experiments for the slot level DM and compared the results with a state-of-the-art approximate POMDP technique and different handcrafted policies. When the number of slot values is very small, the policy computed by approximate POMDP solvers such as Perseus is the best solution (Fig. 6.8). The performance of the DDN-POMDP policy after optimizing its internal reward is close to the state-of-the-art approximate counterpart. For problems with a larger number of slot values, the DDN-POMDP policy works well when tested with different types of errors: the probability of an error in the user's action induced by stress (Fig. 6.10, 6.13), the probabilities of the observation of the user's action and the observation of the user's affective state errors (Fig. 6.11). The DDN-POMDP policy is also robust enough to cope with variation in the user's model (Fig. 6.12).

Chapter 7

Conclusions

As stated in Chapter 1, the central goal of this thesis is to develop a computational model for implementing a robust dialogue manager that is able to adapt its strategies accordingly given observations (with uncertainty) of the user's action and affective state. The thesis has fulfilled this goal by proposing a tractable hybrid DDN-POMDP method that:

- is *robust* to cope with errors from observations of the user's action and affective state;
- is able to take into account the *uncertainty* (i.e., partially observable) and *dynamic aspects* (i.e., change over time) of the user's goal, intention, and the user's affect;
- is *tractable* for real-world frame-based dialogue problems.

Section 7.1 further discusses about the method and summarizes the main points presented in this thesis. Future directions are presented in Section 7.2.

7.1. Summary of the Thesis

Despite the progress in the fields of affect recognition and dialogue system in recent years, design and development of affective dialogue systems pose many research challenges. Computers are still not very good at recognizing speech from the user. Recognizing emotions is even harder for both humans and computers. Extending spoken dialogue systems with multimodal input and output capacities might alleviate the speech recognition error and improve the affect recognition accuracy, but cannot completely solve *uncertainty* factors such as user's goal, user's intention and user's affective state. For example, the user might change their mind (e.g., goal) during the system-user interaction process.

Given the attractiveness of the POMDP theory presented in Section 2.5, I first investigated a factored POMDP approach to modeling affective dialogue (Chap. 5). The proposed dialogue model, called ADS-POMDP, is able to handle the user's goal, user's action, user's affective state and other (user's) hidden components in an elegant manner. We evaluated the performance of the ADS-POMDP policy and compared it with three handcrafted policies and a greedy policy. The experimental results conducted for a single-slot route navigation example demonstrated that the ADS-POMDP policy outperforms the greedy and handcrafted policies (Fig. 5.9). Furthermore, the ADS-POMDP policy also outperforms its spoken counterpart (i.e. SDS-POMDP [179]) when the probability of the user's action error induced by stress p_e is greater than 0 (Fig. 5.7).

To confirm the goodness of the POMDP-based dialogue management model, we further developed the POMDP-based dialogue managers for two real-world dialogue systems: the Ritel system [71] and the Virtual guide system [79]. The results (see Appendix A) showed that the POMDP policy outperforms the optimized handcrafted policy. In addition, we could significantly improve performance of the handcrafted policies by adapting them according to the strategies found by the POMDP policies.

However, we identified several problems with modeling dialogue as a POMDP. First, the definition of a good reward model is a hard problem. The POMDPs proved very sensitive to small change in the reward model, in particular the relative magnitude of different types of reward. Designing a good reward model, therefore, requires both expert knowledge and practical debugging. Second and also the major problem, is the tractability. Most of the current POMDP-based dialogue managers are only restricted themselves to toy frame-based dialogue problems with a few slots and slot values (see Table 5.3).

A recent scaled-up method for spoken dialogue is to compress the POMDP structure [176]. However, the affective dialogue model requires a more complex POMDP structure than that of the spoken counterpart. Compressing the POMDP structure prevents us to incorporate a rich model of the user's affect into the state space and might loose dependencies between the user's emotion, goal, and other hidden state variables.

We therefore proposed a hybrid DDN-POMDP method to handle multi-slot dialogue problems without compressing the POMDP structure (Chap. 6). The method is inspired by the Rapid Dialogue Prototyping Methodology (RDPM) presented in Chapter 3. That is the dialogue manager is divided into two parts: a slot level dialogue manager and a global dialogue manager. Each slot in the first part is modeled as a factored POMDP and approximated as a set of DDNs. The second part is responsible for updating the dialogue information state and selecting the system action. The dialogue manager activity process is explained by a cycle of four steps (Sec. 6.2.3).

To allow for a real-time belief update and an on-line action selection, the DDN-POMDP method only maintains a small number of look-ahead steps (see Sec. 6.2.1). The experimental results showed that the performance of the DDN-POMDP policy outperforms three handcrafted policies when tested with different types of errors: the probability of an error in the user's action induced by stress (Fig. 6.10, 6.13), the probabilities of observed user's action and observed user's affective state errors (Fig. 6.11). The DDN-POMDP policy is also robust enough to cope with variation in the user's model (Fig. 6.12).

However the DDN-POMDP policy is suboptimal when compared with the approximate POMDP policy. This is because the approximate POMDP policy is computed with infinite look-ahead steps. To improve the performance of the DDN-POMDP policy, we have found an elegant way to use short-term reward function to maximize the long-term goal. The idea is to tune the internal reward function to achieve a better policy (Sec. 6.4.2). The performance of the DDN-POMDP policy after optimizing its internal reward is close to the approximate POMDP counterpart (Fig. 6.8). To our knowledge (at least in the dialogue research community), no other work have mentioned this promising solution.

In short, this thesis explored possibilities of applying the POMDPs for modeling affective dialogue. Although a substantive research is still required to design a POMDP-based dialogue model for real-world affective dialogue systems, we belief that this sort of model is beneficial in solving many practical issues in the dialogue management design. A key contribution of this thesis is the proposed DDN-POMDP method that is able to handle frame-based problems with hundreds of slots and hundreds of slot values.

Beside the affective dialogue issue, this thesis makes several contributions to the design and development of traditional frame-based dialogue system. We proposed a RDPM that allows for a quick production of frame-based dialogue models (Chap. 3). Two distinctive features of the methodology are: *supporting* rapid prototype and *facilitating* (spoken) WoZ experiments. The practical result showed that an initial dialogue model for simple applications such as the RestInfo system can be developed in several hours. WoZ interfaces are generated automatically from a WoZ interface generator. These interfaces are integrated with the dialogue management library, which facilitates the tasks of the wizards in a WoZ experiment. The extended version of the methodology was used as a WoZ experiment platform for the INSPIRE project. It is also used for developing several prototype spoken and multimodal prototype dialogue systems (for demonstration purposes). Based on the RDPM, we proposed a *generic dialogue modeling methodology* for the design of multi-application systems (Chap. 4). This has been illustrated in a scenario example of the ICIS project.

Last but not least, in the framework of this thesis, we created several software toolkits that are useful for building practical dialogue systems: (i) the POMDP toolkit for the development of POMDP-based dialogue managers[1]; ii) a prototype DDN-POMDP dialogue management module for the ICIS-CHIM demonstrator[2]; and (iii) the RDPM toolkit (including the Wizard of Oz (WoZ) interface generator) allows for a quick production of frame-based dialogue models and their associated dialogue-driven interfaces.

7.2. Future directions

The DDN-POMDP approach presented in Chapter 6 is a starting point to allow for creating a workable POMDP-based dialogue manager that is able to handle hundreds

[1]http://wwwhome.cs.utwente.nl/∼hofs/pomdp/
[2]http://hmi.ewi.utwente.nl/icis/demonstrator/

of slots with hundreds of slot values. The current version of the global dialogue manager is based on a set of rather simple rules and a small number of global system actions. Therefore, improving this part, such as modeling the action selection component as a POMDP, might help to enhance the performance of the DDN-POMDP dialogue manager as well as to allow the DDN-POMDP dialogue manager to handle other practical issues, which we addressed in Chapter 3, such as a dialogue dead-end situation (i.e. zero solution is retrieved from the database). Our dialogue manager provides a potential framework for investigating tractable factored POMDP algorithms.

Although the current models of user behavior are handcrafted, we believe that they are appropriate for modeling the user affect. Various DBN-based models have been empirically used for modeling the user's affect [56, 65, 100, 101, 191]. It will be interesting to extend our slot POMDP structure, especially by adding more specific features related to the user's affect such as mood and personality, and specifying their dependencies. This extension will help to explain more clearly not only how the user expresses emotions in context but also why they express them.

Another important research topic that is closely related to statistical dialogue management is *user simulation*. Recent work puts emphasis on building the probabilistic user model for spoken dialogue systems from a small dialogue corpus [147]. Design of the affective user simulation needs to incorporate both data from corpora and knowledge about emotions from psychology and other related disciplines. This will be an interesting research avenue to generate artificial corpora for well-defined task environments such as crisis management and use them for training and testing dialogue systems.

Modeling the user's goal for the information seeking dialogue is quite straightforward. We would like to see the extension of the POMDP-based dialogue model for other application domains such as the tutoring domain where no clear user goal can be defined [2]. This extension might help to (partially) answer an open research question posed by *D'Mello et al.* [57]:"How can an affect-sensitive dialogue tutor respond to the learner in a fashion that optimizes learning and engagement?".

Finally, the POMDP-based dialogue management model clearly belongs to the agent-based dialogue model which is the most complex dialogue models [6, 107]. This thesis has restricted itself to applying the model to the frame-based or slot-filling applications. Generally speaking, it is possible to combine the POMDP model with the information state model [165] similar to the hidden information state model [185]. An immediate step is to extend the POMDP-based dialogue model for multi-application systems presented in Chapter 4. That is to say, each node in the dialogue model hierarchy is modeled as a POMDP. These POMDPs will nominate their local actions when they are activated. The root POMDP will determine how to combine these actions and to send a final action to the user. A similar idea (for MDPs) is discussed in *Bohus* [20].

Appendix A

Practical dialogue manager development using POMDPs

This appendix is written based on Bui et al. [33]. It shows several practical issues in applying Partially Observable Markov Decision Processs (POMDPs) for other real-world dialogue problems being developed at the Human Media Interaction Group, University of Twente. The work was conducted in collaboration with Boris van Schooten and Dennis Hofs.

A.1. Introduction

Partially Observable Markov Decision Processs (POMDPs) are attractive for dialogue management in the cases where the Dialogue Manager (DM) has to make choices which depend on statistical information. They can determine optimal strategies in the face of error and partial information. POMDPs can take advantage of statistical information about behavior or error to the fullest extent, and take into account extensive hidden information.

Using POMDPs for spoken dialogue management has been examined thoroughly in *Williams and Young* [177]. Current POMDP-based dialogue managers model a complete slot-filling dialogue including all slots with all values. Large numbers of slots and values lead to a large state space, which is not tractable for current POMDP solvers. Usually, this restricts us to toy problems. Recent efforts to scale up POMDP-based models are reported in *Bui et al.* [32], *Williams and Young* [177].

It is not yet clear enough how to employ POMDPs in a systematic development cycle. A number of practical issues with POMDPs has not really been addressed yet. How do you obtain the user model and the probability distributions? How do you test and debug POMDPs? How do you tweak reward values? How do you evaluate and compare performance of the POMDP policy which other approaches? We address these questions by using the factored POMDP models [32, 177] as a basis, and applying them to two dialogue management systems.

A.2. Methodology

A.2.1. Design guidelines

The state space represents the user's state and action. It is defined as a set of features. We should keep it compact. This can be done by specifying only features which are relevant in selecting the system action and by pruning all unreachable states. For example, when analyzing the Williams's 1945-state travel problem [177], we found that we could increase tractability by pruning 1626 states[1], leaving only 319 reachable states. The system actions are not only the actions toward the user but also actions for other DM tasks such as querying the database. Similar to the state space, the observation space is also defined as a set of observation features such as user's action with noise (from the Automatic Speech Recognition (ASR)) and observed user's emotional state.

Designing a reward model that leads to a good policy is a very challenging task. The typical parameters used to design a reward model are task success, the number of turns, and dialogue act appropriateness (e.g., the system should not confirm a value if it has not yet been provided by the user). The precise numerical values used may have significant impact on the policy and convergence behavior.

A.2.2. Evaluation setup and toolset

From the literature, the typical approach is first to test the quality of the POMDP-based dialogue policy with a simulated user. The real-user evaluation is considered at the final step. An advantage of modeling dialogue as a POMDP is that we can use the POMDP environment model itself as a simulated user model. The probability distributions of the simulated user (testing model) might be varied with the ones of the DM (training model). The probability distributions of all the user models used in our two applications are handcrafted. We have developed a software toolkit to conduct our experiments, which includes a factored POMDP to flat POMDP translator, and an interactive simulator for both the user and the system. The POMDP problem is first solved with a POMDP solver (we used Perseus [161] and ZMDP [157]). The generated alpha file is then used to carry out the performance test with simulated user models. Section A.3 shows our test results on two different problems. We conducted a large number of dialogue episodes ($\geq 10,000$) to guarantee the statistical significance.

A.3. Evaluation

A.3.1. Ritel QA dialogue system.

Ritel [71] is a telephone-based question answering (QA) dialogue system. Dialogue functionality includes confirmation of key phrases and the type of the answer sought, and handling follow-up questions. In our model, we focus on confirmation, modeling key phrases and answer type as slots. In the real system, there are thousands of

[1]For example, the states which the user's goal feature is *ab* and the user's action feature is *c*

possible key phrases, but answer type only has a few possible values. To make it tractable, we simplified the model to one slot with between 3 and 10 values, suitable at least for modeling answer type fully.

The POMDP state space consists of the user goals and the user actions ($S = G_u \times A_u$). The user goals are the different questions or question types that the user may ask, $G_u = q_1, ..., q_n$. The user actions are composed of the questions, plus positive and negative feedback, a 'bye' utterance, and a 'hang-up' signal, $A_u = q_1, ..., q_n, pos, neg, bye, null$. The observation set Z is the same as Au. The system actions consist of confirming each question, answering it, and the 'ask' action, posing an open question to the user, $A = confirm_{q_i}, answer_{q_i}, ask$. When the system answers the correct question, the user poses a new question, otherwise the user either repeats or gives negative feedback. The user may hang up in any dialogue turn, with a fixed probability 0.1.

We made the reward model as simple as possible: give a reward of 1 for answering the right question, -1 for answering a wrong one, zero otherwise. We found that modeling the dialogue state was not necessary, and it increases state space to intractable levels. This model yields the desired behavior, though like *Williams and Young* [177], we found that the system starts confirming even when the user has not yet said anything. This can be remedied by rewarding the *ask* action with a reward slightly more than 0. Note that it is not necessary to give an explicit penalty for dialogue length. The problem can be translated as: answer as many questions as possible before the user hangs up. The results of Perseus were not useable, so the experiments were done with ZMDP only. Convergence was good up till nine slot values. We observed that, when the ASR error becomes high, 0.7 or above, the system actually wants to hear a question multiple times in a row before answering it. The policy was compared to a hand-crafted policy (Fig. A.1), similar to the actual Ritel policy, which is based on counting the number of times a particular keyword was heard. It was optimized to each particular problem by determining the optimal number of times a question should be heard before confirmation is sufficient, just as the POMDP does.

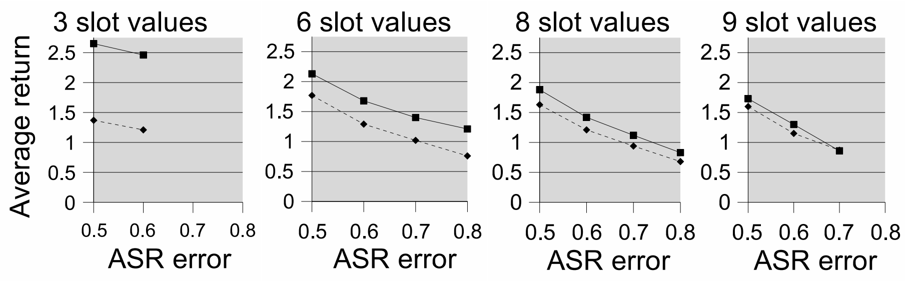


Figure A.1: Performance comparison of POMDP and optimised hand-crafted models for different problem sizes and ASR error rates. The solid line is the POMDP, the dashed line is the hand-crafted model. For three values, an error more than 0.6 would result in the probability of hearing the wrong question being higher than the right one. For nine values and error=0.8, no sensible policy could be calculated.

A.3.2. Virtual Guide application.

The Virtual Guide is a character in a Virtual Reality model of the Music Centre in Enschede [79]. The character can help users find their way in the building. It encompasses a multimodal dialogue system that allows users to refer to locations and objects with spoken or written language or by pointing at a location on a map. The system uses clarification questions and implicit confirmations. The user can continue a dialogue with follow-up questions.

It is currently impossible to create a tractable POMDP model for the system. In our simplified models the user can only ask for the route between two objects, and the world is limited to three or eight objects. Moreover we made a closed model where follow-up questions are not allowed. We fitted the problem into the SDS-POMDP dialogue model [177]. A reward is given when the system gives the correct route and the user provided both locations.

Evaluations were performed for four models. For each of them we compared the solutions of Perseus and ZMDP and an adapted hand-crafted system. The solvers were run until convergence was usually slowing down (about ten minutes), although it had not always reached a desirable level. We then ran dialogues with an automatic user simulation based on the user model of the POMDP.

The first model stops after giving any answer, has observation error 0.2, and three locations. We varied the observation error of the simulator. The results in Figure A.2 show that with increasing errors the POMDP solutions produced higher returns than the hand-crafted system.

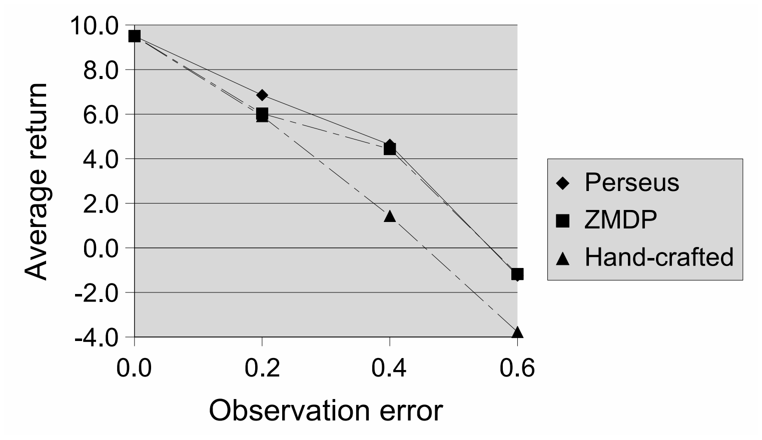


Figure A.2: Average returns for simulation with different observation errors

For the second model, we increased the observation error to 0.6. The Perseus solution contained a state from which the dialogue never ended. ZMDP did not converge acceptably. Therefore its solution performed worse than the hand-crafted system.

The third model has observation error 0.2 again, but the dialogue only stops after giving a correct answer. The average returns for Perseus, ZMDP and the hand-crafted system were 8.08, 6.84 and 6.69 (higher than the first model, because of a reward for an extra system action).

In the last model we increased the number of locations to eight, resulting in 729 POMDP states instead of about 80. Perseus was not able to load this problem. The average returns obtained with ZMDP and the hand-crafted system were 5.08 and 4.04.

A.4. Conclusions

Although our experiments indicate that POMDP-based dialogue systems can perform better than hand-crafted ones, we identified several problems with modeling them. One of the major problems remains tractability. It is not possible to obtain useful solutions for any but strongly simplified models, which may bear little relation to the original problem. For example, when reducing the number of slot values, the strategy of trying them one by one can be employed, something that may not have been feasible for the original number of values. Another example was the need to simplify an open model, where the end of a dialogue is determined by the user, to a closed model.

The definition of a good reward model is another hard problem. While the reward model models psychological factors such as user satisfaction, which cannot easily be quantified precisely, the POMDPs proved very sensitive to small changes in the reward model, in particular the relative magnitude of different types of reward. In practice we had to experiment with different reward values.

The POMDP policies sometimes came up with surprising strategies. For example, some policies decided to confirm multiple times in a row, something which our original hand-crafted models did not. We could significantly improve performance of the hand-crafted policies by adapting them according to the strategies found by the POMDP policies. This shows how POMDPs could be used to improve hand-crafted systems.

Appendix B

Example interaction

B.1. Single slot

Table B.1 shows three representative dialogue episodes of the interaction between the DDN-POMDP dialogue manager and the simulated user for a 1-slot case. In all dialogue episodes, the prior probability distributions of Gu and Eu are equally distributed. The initial state of Du is *notstated*. Based on these probability distributions and the initial state, the dialogue manager selects the *ask* action in its first turn. The hidden part is the information generated by the simulated user and it is unobservable to the dialogue manager. The example illustrates some adaptive behavior of the DDN-POMDP dialogue manager such as it chooses the *fail* action when $oeu = extreme$ (dialogue episode 1), the *ok* action when $oeu = no$ or $oeu = low$ (dialogue episode 3), the *confirm* action when $oeu = moderate$ or $oeu = high$ (dialogue episode 2). The average time for each turn is about 93 ms.

B.2. Ten slots

Table B.2 illustrates the interaction between the DDN-POMDP prototype dialogue manager and the author of this thesis. The prior probability distributions of 10 slot-POMDP parameter files are equally distributed, therefore all slots nominate the *ask* action, the AS applies the first rule specified in Section 6.3.2 (rule 1) and the $ask(open)$ action is selected. Rules 4, 5, 5, and 3 are used in S_2, S_3, S_4 and S_5, respectively.

Dialogue episode 1	
S_1 :	Please provide the location of your team members? [*ask*]
U_1 :	Building 6
	$oau{=}v_6; oeu{=}extreme; r{=}$-1.0 $[gu{=}v_6, eu{=}extreme, au{=}v_6, du{=}stated]$
S_2 :	Please wait a moment, I will connect you through our route planning team. [*fail*]
	End of the episode: episodeReward=-4.0
Dialogue episode 2	
S_1 :	Please provide the location of your team members? [*ask*]
U_1 :	Building 9
	$oau{=}v_9; oeu{=}high; r{=}$-1.0 $[gu{=}v_9, eu{=}high, au{=}v_9, du{=}stated]$
S_2 :	I understood Building 9. Is that correct? $[confirm(v_9)]$
U_2 :	Yes
	$oau{=}yes; oeu{=}high; r{=}$-1.0 $[gu{=}v_9, eu{=}high, au{=}yes, du{=}stated]$
S_3 :	Ok, the route description is shown on your PDA. $[ok(v_9)]$
	End of the episode: episodeReward=8.0
Dialogue episode 3	
S_1 :	Please provide the location of your team members? [*ask*]
U_1 :	Building 7
	$oau{=}v_7; oeu{=}low; r{=}$-1.0 $[gu{=}v_7, eu{=}low, au{=}v_7, du{=}stated]$
S_2 :	Ok, the route description is shown on your PDA. $[ok(v_7)]$
	End of the episode: episodeReward=9.0
	$\{v_1 = Bakkerskade, v_2 = Bakkersweg, v_3 = Propaanweg, v_4 = Buttaanweg, v_5 = Beneluxweg, v_6 = Building\ 6, v_7 = Building\ 7, v_8 = Building\ 8, v_9 = Building\ 9, v_{10} = Building\ 10\}$

Table B.1: Representative dialogue example for a 1-slot case

S_0 :	What can I do for you?
U_0 :	I need a route description to coordinate with my team members. They are currently located at 10 different locations.
S_1 :	What are these locations? $[rule\ 1 : ask(open)]$
U_1 :	first location is Bakkerskade, second one is Bakkersweg $[oau{=}answer(f_1{=}v_1, f_2{=}v_2), oeu{=}high]$
S_2 :	So, the first location is Bakkerskade and the second location is Bakkersweg. Is that correct? $[rule\ 4 : confirm(f_1{=}v_1, f_2{=}v_2)]$
U_2 :	yes $[oau{=}yes, oeu{=}moderate]$
S_3 :	What about the third location? $[rule\ 5 : ask(f_3)]$
U_3 :	The third, fourth, and fifth locations are: Propaanweg, Buttaanweg, and Beneluxweg $[answer(f_3{=}v_3, f_4{=}v_4, f_5{=}v_5), oeu{=}low]$
S_4 :	And what is the sixth location? $[rule\ 5 : ask(f_6)]$
U_4 :	The remaining locations are buildings six, seven, eight, nine, ten. $[answer(f_6{=}v_6, f_7{=}v_7, f_8{=}v_8, f_9{=}v_9, f_{10}{=}v_{10}), oeu{=}no]$
S_5 :	Ok, the route description is shown on your PDA. $[rule\ 3 : giveSolution(f_1{=}v_1, f_2{=}v_2, f_3{=}v_3, f_4{=}v_4, f_5{=}v_5, f_6{=}v_6, f_7{=}v_7, f_8{=}v_8, f_9{=}v_9, f_{10}{=}v_{10})]$
	$\{v_1 = Bakkerskade, v_2 = Bakkersweg, v_3 = Propaanweg, v_4 = Buttaanweg, v_5 = Beneluxweg, v_6 = Building\ 6, v_7 = Building\ 7, v_8 = Building\ 8, v_9 = Building\ 9, v_{10} = Building\ 10\}$

Table B.2: Dialogue example for a 10-slot case

Bibliography

[1] Aberdeen, D., A (revised) survey of approximate methods for solving partially observable Markov decision processes, *Tech. rep.*, National ICT Australia, 2003.

[2] Ai, H., and D. J. Litman, Knowledge consistent user simulations for dialog systems, in *Proceedings of the 8th Annual Conference of the International Speech Communication Association (INTERSPEECH '07)*, pp. 2697–2700, Antwerp, Belgium, 2007.

[3] Ai, H., and F. Weng, User simulation as testing for spoken dialog systems, in *Proceedings of the 9th SIGDial Workshop on Discourse and Dialogue (SIGdial '08)*, edited by D. Schlangen and B. A. Hockey, pp. 164–171, Columbus, Ohio, USA, 2008.

[4] Ailomaa, M., M. Melichar, A. Lisowska, M. Rajman, and S. Armstrong, Archivus: A multimodal system for multimedia meeting browsing and retrieval, in *Proceedings of the COLING/ACL on Interactive presentation sessions (COLING/ACL '06)*, edited by J. Curran, pp. 49–52, Sydney, Australia, 2006.

[5] Allen, J. F., The TRAINS project: A case study in building a conversational planning agent, *Journal of Experimental and Theoretical AI (JETAI)*, *7*, 7–48, 1995.

[6] Allen, J. F., D. K. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent, Toward conversational human-computer interaction., *AI Magazine*, *22*(4), 27–38, 2001.

[7] André, E., L. Dybkjær, W. Minker, and P. Heisterkamp (Eds.), *Affective Dialogue Systems, Tutorial and Research Workshop (ADS 2004), Kloster Irsee, Germany, June 14-16, 2004, Proceedings*, *Lecture Notes in Computer Science*, vol. 3068, Springer, 2004.

[8] André, E., M. Rehm, W. Minker, and D. Bühler, Endowing spoken language dialogue systems with emotional intelligence, in *Proceedings of ADS 2004*, pp. 178–187, 2004.

[9] Ang, J., R. Dhillon, A. Krupski, E. Shriberg, and A. Stolcke, Prosody-based automatic detection of annoyance and frustration in human-computer dialog, in

Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP-2002), pp. 2037–2040, Denver, Colorado, USA, 2002.

[10] Astrom, K. J., Optimal control of Markov processes with incomplete state information, *Journal of Mathematical Analysis and Applications*, *10*, 174–205, 1965.

[11] Austin, J. L., *How to Do Things with Words*, Harvard University Press, 1962.

[12] Bahar, R. I., E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, Algebraic decision diagrams and their applications, in *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD '93)*, pp. 188–191, IEEE Computer Society Press, Los Alamitos, CA, USA, 1993.

[13] Ball, E., A Bayesian heart: Computer recognition and simulation of emotion, in *Emotions in Humans and Artifacts*, edited by P. P. Robert Trappl and S. Payr, chap. 11, pp. 303–332, The MIT Press, 2003.

[14] Ball, G., and J. Breese, Relating personality and behavior: posture and gestures, in *Affective Interactions: Towards a New Generation of Computer Interfaces*, *Lecture Notes in Computer Science*, vol. 1814/2000, edited by A. Paiva, pp. 196–203, Springer Berlin/Heidelberg, 2000.

[15] Batliner, A., K. Fischer, R. Huber, J. Spilker, and E. Nöth, How to find trouble in communication, *Speech Communication*, *40*(1-2), 117–143, 2003.

[16] Becker, C., N. Lessmann, S. Kopp, and I. Wachsmuth, Connecting feelings and thoughts - modeling the interaction of emotion and cognition in embodied agents, in *Proceedings of the 7th International Conference on Cognitive Modeling (ICCM '06)*, edited by D. Fum, F. Del Missier, and A. Stocco, pp. 32–37, Trieste, Italy, 2006.

[17] Bhatt, K., S. Argamon, and M. Evens, Hedged responses and expressions of affect in human/human and human/computer tutorial interactions, in *Proceedings of the 26th Annual Conference of the Cognitive Science Society (CogSci '04)*, edited by K. Forbus, D. Gentner, and T. Regier, pp. 114–119, Chicago, Illinois, USA, 2004.

[18] Bilange, E., *Dialogue personne-machine : modõlisation et rõalisation informatique*, Hermès, Paris, 1992.

[19] Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd, GUS, a frame-driven dialog system, *Artificial Intelligence*, *8*, 155–173, 1977.

[20] Bohus, D., Error awareness and recovery in task-oriented spoken dialogue systems - PhD thesis proposal, *Tech. rep.*, Carnegie Mellon University, 2004.

[21] Bohus, D., and A. Rudnicky, Ravenclaw: Dialog management using hierarchical task decomposition and an expectation agenda, in *Proceedings of the 8th European Conference on Speech Communication and Technology (EUROSPEECH '03)*, pp. 597–600, Geneva, Switzerland, 2003.

[22] Boutilier, C., and D. Poole, Computing optimal policies for partially observable decision processes using compact representations, in *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI '96)*, vol. 2, pp. 1168–1175, Portland, Oregon, USA, 1996.

[23] Boyce, S., and A. Gorin, User interface issues for natural spoken dialog systems, in *Proceedings of the International Symposium on Spoken Dialogue (ISSD)*, pp. 65–68, Philadelphia, USA, 1996.

[24] Brooks, A., A. Makarenkoa, S. Williamsa, and H. Durrant-Whytea, Parametric POMDPs for planning in continuous state spaces, *Robotics and Autonomous Systems*, *54*(11), 887–897, 2006.

[25] Bui, T. D., D. Heylen, M. Poel, and A. Nijholt, ParleE: An adaptive plan based event appraisal model of emotions, in *KI 2002: Advances in Artificial Intelligence, 25th Annual German Conference on AI, KI 2002 Aachen, Germany, September 16-20, 2002 Proceedings*, *Lecture Notes in Computer Science*, vol. 2479/2002, edited by M. Jarke, J. Koehler, and G. Lakemeyer, pp. 1–9, Springer Berlin/Heidelberg, 2002.

[26] Bui, T. H., Multimodal dialogue management - State of the art, *Tech. rep.*, University of Twente, 2006.

[27] Bui, T. H., and M. Rajman, Rapid dialogue prototyping methodology, *Tech. Rep. IC/2004/01*, Ecole Polytechnique Federale de Lausanne, 2004.

[28] Bui, T. H., M. Rajman, and M. Melichar, Rapid dialogue prototyping methodology, in *Text, Speech and Dialogue: 7th International Conference, TSD 2004, Brno, Czech Republic, September 8-11, 2004, Proceedings*, *Lecture Notes in Computer Science*, vol. 3206/2004, edited by P. Sojka, I. Kopecek, and K. Pala, pp. 579–586, Springer Berlin/Heidelberg, 2004.

[29] Bui, T. H., M. Rajman, and S. Quarteroni, Extending the rapid dialogue prototyping methodology: User modeling, *Tech. Rep. WP4 Deliverable 4.3*, Ecole Polytechnique Federale de Lausanne, 2004.

[30] Bui, T. H., J. Zwiers, A. Nijholt, and M. Poel, Generic dialogue modeling for multi-application dialogue systems, in *Machine Learning for Multimodal Interaction: Second International Workshop, MLMI 2005, Edinburgh, UK, July 11-13, 2005, Revised Selected Papers*, *Lecture Notes in Computer Science*, vol. 3869/2006, edited by S. Renals and S. Bengio, pp. 174–186, Springer Berlin/Heidelberg, 2006.

[31] Bui, T. H., J. Zwiers, M. Poel, and A. Nijholt, Toward affective dialogue modeling using partially observable Markov decision processes, in *Proceedings of Workshop Emotion and Computing, 29th Annual German Conference on Artificial Intelligence*, edited by D. Reichardt, P. Levi, and J.-J. Meyer, pp. 47–50, Bremen, Germany, 2006.

[32] Bui, T. H., M. Poel, A. Nijholt, and J. Zwiers, A tractable DDN-POMDP approach to affective dialogue modeling for general probabilistic frame-based dialogue systems, in *Proceedings of the 5th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (KRPD '07)*, edited by D. Traum, J. Alexandersson, A. Jonsson, and I. Zukerman, pp. 34–37, Hyderabad, India, 2007.

[33] Bui, T. H., B. van Schooten, and D. Hofs, Practical dialogue manager development using POMDPs, in *Proceedings of the 8th SIGdial Workshop on Discourse and Dialogue (SIGdial '07)*, edited by S. Keizer, H. Bunt, and T. Paek, pp. 215–218, Antwerp, Belgium, 2007.

[34] Bui, T. H., M. Poel, A. Nijholt, and J. Zwiers, A tractable hybrid DDN-POMDP approach to affective dialogue modeling for probabilistic frame-based dialogue systems, *Natural Language Engineering*, 2008 (accepted for publication).

[35] Bunt, H. C., Context and dialogue control, *THINK Quarterly*, *3*, 19–31, 1994.

[36] Cahn, J. E., and S. E. Brennan, A psychological model of grounding and repair in dialog, in *Working Papers of the AAAI Fall Symposium on Psychological Models of Communication in Collaborative Systems*, edited by S. E. Brennan, A. Giboin, and D. Traum, pp. 25–33, AAAI Press, Sea Cliff, Massachusetts, USA, 1999.

[37] Carletta, J., A. Isard, S. Isard, J. Knowtko, G. Doherty-Seddon, and A. Anderson, HCRC dialogue structure coding manual, *Tech. rep.*, University of Edinburgh, 1996.

[38] Carofiglio, V., F. de Rosis, and R. Grassano, Dynamic models of mixed emotion activation, in *Animating expressive characters for social interactions*, edited by D. Canamero and R. Aylett, 2003.

[39] Cassandra, A. R., Exact and approximate algorithms for partially observable markov decision processes, Ph.D. thesis, Brown University, 1998.

[40] Cassandra, A. R., L. P. Kaelbling, and M. L. Littman, Acting optimally in partially observable stochastic domains, in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI '94)*, vol. 2, pp. 1023–1028, AAAI Press, Seattle, Washington, USA, 1994.

[41] Cassandra, A. R., M. L. Littman, and N. L. Zhang, Incremental pruning: A simple, fast, exact method for partially observable markov decision processes,

in *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI '97)*, edited by D. Geiger and P. Shenoy, pp. 54–61, Morgan Kaufmann, San Francisco, CA, 1997.

[42] Cassell, J., T. W. Bickmore, M. Billinghurst, L. Campbell, K. Chang, H. H. Vilhjálmsson, and H. Yan, Embodiment in conversational interfaces: Rea, in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit (CHI-99)*, pp. 520–527, ACM, Pittsburgh, Pennsylvania, US, 1999.

[43] Cenek, P., M. Melichar, and M. Rajman, A framework for rapid multimodal application design, in *Text, Speech and Dialogue: 8th International Conference, TSD 2005, Karlovy Vary, Czech Republic, September 12-15, 2005. Proceedings*, *Lecture Notes in Computer Science*, vol. 3658/2005, pp. 393–403, Springer Berlin/Heidelberg, 2005.

[44] Cheng, H.-T., Algorithms for partially observable Markov decision processes, Ph.D. thesis, University of British Columbia, 1988.

[45] Chu-Carroll, J., Form-based reasoning for mixed-initiative dialogue management in information-query systems, in *Proceedings of the 6th European Conference on Speech Communication and Technology (EURSPEECH 99)*, pp. 1519–1522, Budapest, Hungary, 1999.

[46] Chu-Carroll, J., and B. Carpenter, Vector-based natural language call routing, *Computational Linguistics*, *25*(3), 361–388, 1999.

[47] Clark, H. H., and E. F. Schaefer, Contributing to discourse, *Cognitive Science*, *13*, 259–294, 1989.

[48] Cohen, P. R., M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow, QuickSet: Multimodal interaction for distributed applications, in *Proceedings of the 5th ACM International Conference on Multimedia (MULTIMEDIA-97)*, pp. 31–40, ACM, Seattle, Washington, US, 1997.

[49] Colby, K. M., S. Weber, and F. D. Hilf, Artificial paranoia, *Artificial Intelligence*, *2*, 1–25, 1971.

[50] Conati, C., Probabilistic assessment of user's emotions in educational games, *Applied Artificial Intelligence*, *16*(7-8), 555–575, 2002.

[51] Cooper, R., S. Larsson, C. Matheson, M. Poesio, and D. R. Traum, Coding instructional dialogue for information states, *Tech. rep.*, G oteborg University, 1999.

[52] Cutting, D. R., D. R. Karger, J. O. Pedersen, and J. W. Tukey, Scatter/gather: a cluster-based approach to browsing large document collections, in *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR-92)*, edited by N. Belkin, P. Ingwersen, and A. M. Pejtersen, pp. 318–329, ACM, Copenhagen, Denmark, 1992.

[53] Dahlbäck, N., Towards a dialogue taxonomy, in *Dialogue Processing in Spoken Language Systems: ECAI'96 Workshop Budapest, Hungary, August 13, 1996 Revised Papers*, *Lecture Notes in Computer Science*, vol. 1236/1997, edited by E. Maier, M. Mast, and S. LuperFoy, pp. 29–40, Springer Berlin/Heidelberg, 1997.

[54] Dahlbäck, N., A. Jönsson, and L. Ahrenberg, Wizard of Oz studies - why and how, *Knowledge Based Systems*, *6*(4), 258–266, 1993.

[55] Daly-Jones, O., N. Bevan, and C. Thomas, Wizard-of-Oz prototyping, in *Handbook of User-Centred Design, http://www.ejeisa.com/nectar/inuse/6.2/3-3.htm*, 1999.

[56] de Rosis, F., N. Novielli, V. Carofiglio, A. Cavalluzzi, and B. D. Carolis, User modeling and adaptation in health promotion dialogs with an animated character, *Journal of Biomedical Informatics*, *39*(5), 514–531, 2006.

[57] D'Mello, S., R. W. Picard, and A. Graesser, Toward an affect-sensitive autotutor, *IEEE Intelligent Systems*, *22*(4), 53–61, 2007.

[58] Eckert, W., E. Levin, and R. Pieraccini, User modelling for spoken dialogue system evaluation, in *Prococeedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU-97)*, pp. 80–87, IEEE, Santa Barbara, California, 1997.

[59] El-Nasr, M. S., J. Yen, and T. R. Ioerger, Flame - fuzzy logic adaptive model of emotions, *Autonomous Agents and Multi-Agent Systems*, *3*(3), 219–257, 2000.

[60] Elliott, C., J. Rickel, and J. C. Lester, Lifelike pedagogical agents and affective computing: An exploratory synthesis, in *Artificial Intelligence Today: Recent Trends and Developments*, *Lecture Notes in Computer Science*, vol. 1600/1999, edited by M. J. Wooldridge and M. Veloso, pp. 195–211, Springer Berlin/Heidelberg, 1999.

[61] Elliott, C. D., The affective reasoner: A process model of emotions in a multi-agent system, Ph.D. thesis, Northwestern University, 1992.

[62] Feng, Z., and S. Zilberstein, Region-based incremental pruning for pomdps, in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI '04)*, edited by M. Chickering and J. Halpern, pp. 146–153, AUAI Press, Banff, Canada, 2004.

[63] Feng, Z., and S. Zilberstein, Efficient maximization in solving pomdps, in *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI '05)*, edited by M. M. Veloso and S. Kambhampati, pp. 975–980, AAAI Press, Pittsburgh, Pennsylvania, USA, 2005.

[64] Ferguson, G., and J. F. Allen, TRIPs: an integrated intelligent problem-solving assistant, in *Proceedings of the 15th National Conference on Artificial intelligence (AAAI '98)*, pp. 567–572, AAAI Press, Menlo Park, CA, USA, 1998.

[65] Fernandez, R., and R. W. Picard, Modeling drivers' speech under stress, *Speech Commununication*, *40*(1-2), 145–159, 2003.

[66] Fitrianie, S., et al., A multimodal human-computer interaction framework for research into crisis management, in *Proceedings of the fourth international conference on information systems for crisis management (ISCRAM '07)*, pp. 149–158, Academic and Scientific Publishers NV, 2007.

[67] Flycht-Eriksson, A., A survey of knowledge source in dialogue systems, in *Proceedings of 1st IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (KRPD '99)*, Stockholm, Sweden, 1999.

[68] Forler, E., Intelligent user interface for specialized web sites, Master's thesis, Ecole Polytechnique Federale de Lausanne, 2000.

[69] Foster, M. E., State of the art review: Multimodal fission, *Tech. rep.*, University of Edinburgh, 2002.

[70] Fraser, N. M., and G. N. Gilbert, Simulating Speech Systems, *Computer Speech and Language*, *3-5*, 1991.

[71] Galibert, O., G. Illouz, and S. Rosset, Ritel: an open-domain, human-computer dialog system, in *Proceedings of the 8th Annual Conference of the International Speech Communication Association (INTERSPEECH '05)*, pp. 909–912, Lisbon, Portugal, 2005.

[72] Geutner, P., F. Steffens, and D. Manstetten, Design of the VICO spoken dialogue system: Evaluation of user expectations by wizard-of-oz experiments, in *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC '02)*, Las Palmas, Spain, 2002.

[73] Goddeau, D., H. Meng, J. Polifroni, S. Seneff, and S. Busayapongchaiy, A form-based dialogue manager for spoken language applications, in *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP '96)*, vol. 2, pp. 701–704, Philadelphia, PA, 1996.

[74] Gratch, J., and S. Marsella, A domain-independent framework for modeling emotion, *Cognitive Systems Research*, *5-4*, 269–306, 2004.

[75] Hansen, B., D. G. Novick, and S. Sutton, Systematic design of spoken prompts, in *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground (CHI 96)*, edited by M. J. Tauber, pp. 157–164, ACM, Vancouver, Canada, 1996.

[76] Hauskrecht, M., Value-function approximations for partially observable Markov decision processes, *Journal of Artificial Intelligence Research (JAIR)*, *13*, 33–94, 2000.

[77] Heylen, D., A. Nijholt, and R. op den Akker, Affect in tutoring dialogues, *Applied Artificial Intelligence*, *19*, 287–311, 2005.

[78] Hoey, J., A. von Bertoldi, P. Poupart, and A. Mihailidis, Assisting persons with dementia during handwashing using a partially observable markov decision process, in *Proceedings of the 5th International Conference on Vision Systems (ICVS '07)*, Bielefeld, Germany, 2007.

[79] Hofs, D., R. op den Akker, and A. Nijholt, A generic architecture and dialogue model for multimodal interaction, in *Proceedings of the 1st Nordic Symposium on Multimodal Communication*, edited by P. Paggio, K. Jokinen, and A. Jönsson, pp. 79–92, Copenhagen, Denmark, 2003.

[80] Holzapfel, H., C. Fuegen, M. Denecke, and A. Waibel, Integrating emotional cues into a framework for dialogue management, in *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces (ICMI '02)*, pp. 141–146, Pittsburgh, Pennsylvania, USA, 2002.

[81] Howard, R. A., *Dynamic Programming and Markov Process*, The MIT Press, 1960.

[82] Huber, R., A. Batliner, J. Buckow, E. Nöth, V. Warnke, and H. Niemann, Recognition of emotion in a realistic dialogue scenario, in *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP 2000)*, vol. 1, pp. 665–668, Beijing, China, 2000.

[83] Hulstijn, J., R. Steetskamp, H. ter Doest, S. van de Burgt, and A. Nijholt, Topics in SCHISMA dialogues, in *Proceedings of the Twente Workshop on Language Technology 11 (TWLT 11)*, edited by S. LuperFoy, A. Nijholt, and G. V. van Zanten, pp. 89–99, Enschede, The Netherlands, 1996.

[84] Jain, A., M. Murty, and P. Flynn, Data clustering: a review, *ACM Computing Surveys*, *31*(3), 264–323, 1999.

[85] Johnson, S. C., Hierarchical clustering schemes, *Psychometrika*, *32*(3), 241–254, 1967.

[86] Johnston, M., Unification-based multimodal parsing, in *Proceedings of the 36th annual meeting on Association for Computational Linguistics (ACL 98)*, vol. 1, pp. 624–630, ACL, Montreal, Quebec, Canada, 1998.

[87] Johnston, M., S. Bangalore, G. Vasireddy, A. Stent, P. Ehlen, M. A. Walker, S. Whittaker, and P. Maloor, MATCH: An architecture for multimodal dialogue systems, in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL-02)*, pp. 376–383, ACL, Philadelphia, Pennsylvania, USA, 2002.

[88] Jurafsky, D., and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing*, Prentice Hall, Englewood Cliffs, NJ, 2000.

[89] Kaelbling, L. P., M. L. Littman, and A. R. Cassandra, Planning and acting in partially observable stochastic domains, *Artificial Intelligence*, *101*(1-2), 99–134, 1998.

[90] Kanazawa, K., and T. Dean, A model for projection and action, in *Proceedings of the 11th International Joint Conferences on Artificial Intelligence (IJCAI '89)*, vol. 2, pp. 985–990, Morgan Kaufmann, Detroit, MI, USA, 1989.

[91] Katsionis, G., and M. Virvou, A cognitive theory for affective user modelling in a virtual reality educational game, in *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics (SMC '04)*, vol. 2, pp. 1209–1213, IEEE, The Hague, The Netherlands, 2004.

[92] Kim, D., H. S. Sim, K.-E. Kim, J. H. Kim, H. Kim, and J. W. Sung, Effects of user modeling on POMDP-based dialogue systems, in *Proceedings of INTER-SPEECH '08*, Brisbane, Australia, 2008 (to appear).

[93] Kort, B., R. Reilly, and R. W. Picard, An affective model of interplay between emotions and learning: Reengineering educational pedagogy-building a learning companion, in *Proceedings of the 2nd IEEE International Conference on Advanced Learning Technologies (ICALT 2001)*, pp. 43–46, IEEE Computer Society, Madison, WI, USA, 2001.

[94] Larsson, S., Issued-based dialogue management, Ph.D. thesis, Göteborg University, Sweden, 2002.

[95] Larsson, S., and D. R. Traum, Information state and dialogue management in the TRINDI dialogue move engine toolkit, *Natural Language Engineering*, *6*(3-4), 323–340, 2000.

[96] Lee, C. M., and S. S. Narayanan, Toward detecting emotions in spoken dialogs, *IEEE Transactions on Speech and Audio Processing*, *13-2*, 293–303, 2005.

[97] Levelt, W. J., *Speaking: From Intention to Articulation*, The MIT Press, 1989.

[98] Levin, E., R. Pieraccini, and W. Eckert, A stochastic model of human-machine interaction for learning dialogue strategies, *IEEE Transactions on Speech and Audio Processing*, *8(1)*, 11–23, 2000.

[99] Levinson, S. C., *Pragmatics*, Cambridge University Press, 1983.

[100] Li, X., and Q. Ji, Active affective state detection and user assistance with dynamic bayesian networks, *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *35*(1), 93–105, 2005.

[101] Liao, W., W. Zhang, Z. Zhu, Q. Ji, and W. D. Gray, Toward a decision-theoretic framework for affect recognition and user assistance, *International Journal of Man-Machine Studies*, *64*(9), 847–873, 2006.

[102] Lisowska, A., M. Rajman, and T. H. Bui, Archivus: A system for accessing the content of recorded multimodal meetings, in *Machine Learning for Multimodal Interaction, First International Workshop,MLMI 2004, Martigny, Switzerland, June 21-23, 2004, Revised Selected Papers*, vol. 3361, edited by S. Bengio and H. Bourlard, pp. 291 – 304, Springer, 2004.

[103] Littman, M. L., A. R. Cassandra, and L. P. Kaelbling, Learning policies for partially observable environments: Scaling up, in *Proceedings of the 12th International Conference on Machine Learning (ICML-95)*, edited by A. Prieditis and S. J. Russell, pp. 362–370, Morgan Kaufmann, Tahoe City, California, USA, 1995.

[104] Martinho, C., I. Machado, and A. Paiva, A cognitive approach to affective user modeling, in *Affective Interactions: Towards a New Generation of Computer Interfaces*, *Lecture Notes in Computer Science*, vol. 1814/2000, edited by A. Paiva, pp. 64–75, Springer Berlin/Heidelberg, 2000.

[105] Martinovsky, B., and D. R. Traum, The error is the clue: Breakdown in human-machine interaction, in *Proceedings of the ISCA Tutorial and Research Workshop on Error handling in Spoken Dialogue Systems (EHSD '03)*, pp. 11–16, Château d'Oex, Vaud, Switzerland, 2003.

[106] McTear, M., Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit, in *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP-98)*, pp. 1223–1226, Sydney, Australia, 1998.

[107] McTear, M., Spoken dialogue technology: Enabling the conversational user interface, *ACM Computing Survey, Volume 34, No 1*, 2002.

[108] McTear, M. F., *Spoken dialogue technology: toward the conversational user interface*, Springer Verlag, 2004.

[109] Melichar, M., Design of multimodal dialogue-based systems, Ph.D. thesis, Ecole Polytechnique Federale de Lausanne, 2008.

[110] Melichar, M., and P. Cenek, From vocal to multimodal dialogue management, in *Proceedings of the 8th international conference on Multimodal interfaces (ICMI '06)*, pp. 59–67, ACM, Banff, Alberta, Canada, 2006.

[111] Minsky, M., A framework for representing knowledge, *Tech. rep.*, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, 1974.

[112] Monahan, G. E., A survey of partially observable markov decision processes: Theory, models, and algorithms, *Management Science*, *28-1*, 1–16, 1982.

[113] Murphy, K. P., A survey of POMDP solution techniques, *Tech. rep.*, University of California, Berkeley, USA, 2000.

[114] Nasoz, F., and C. L. Lisetti, Affective user modeling for adaptive intelligent user interfaces, in *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments: 12th International Conference, HCI International 2007, Beijing, China, July 22-27, 2007, Proceedings, Part III*, *Lecture Notes in Computer Science*, vol. 4552/2007, edited by J. A. Jacko, pp. 421–430, Springer Berlin/Heidelberg, 2007.

[115] Nijholt, A., D. Reidsma, H. van Welbergen, H. op den Akker, and Z. Ruttkay, Mutually coordinated anticipatory multimodal interaction, in *Nonverbal Features of Human-Human and Human-Machine Interaction*, Lecture Notes in Computer Science, pp. 73–93, Springer Verlag, Berlin, 2008.

[116] Ortony, A., G. L. Clore, and A. Collins, *The Cognitive Structure of Emotions*, Cambridge University Press, 1988.

[117] Oviatt, S., Multimodal interfaces, in *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, edited by J. A. Jacko and A. Sears, pp. 286–304, Lawrence Erlbaum Assoc., Mahwah, NJ, 2003.

[118] Paek, T., and E. Horvitz, Conversation as action under uncertainty, in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI '00)*, pp. 455–464, Morgan Kaufmann, San Francisco, CA, USA, 2000.

[119] Paek, T., and E. Horvitz, Grounding criterion: Toward a formal theory of grounding, *Tech. Rep. MSR-TR-2000-40*, Microsoft Research, 2000.

[120] Parr, R., and S. J. Russell, Approximating optimal policies for partially observable stochastic domains, in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI '95)*, vol. 2, pp. 1088–1095, Morgan Kaufmann, Montréal, Québec, Canada, 1995.

[121] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Francisco, CA, USA, 1988.

[122] Pelachaud, C., V. Carofiglio, B. D. Carolis, F. de Rosis, and I. Poggi, Embodied contextual agent in information delivering application, in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '02)*, vol. 2, pp. 758–765, ACM, Bologna, Italy, 2002.

[123] Picard, R. W., *Affective Computing*, The MIT Press, 1997.

[124] Picard, R. W., E. Vyzas, and J. Healey, Toward machine emotional intelligence: Analysis of affective physiological state, *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *23*(10), 1175–1191, 2001.

[125] Pieraccini, R., and J. Huerta, Where do we go from here? research and commercial spoken dialog systems, in *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue (SIGdial '05)*, edited by L. Dybkjær and W. Minker, pp. 1–10, Lisbon, Portugal, 2005.

[126] Pietquin, O., A framework for unsupervised learning of dialogue strategies, Ph.D. thesis, Universitaires de Louvain, 2004.

[127] Pineau, J., N. Roy, and S. Thrun, A hierarchical approach to pomdp planning and execution, in *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)*, 2001.

[128] Pineau, J., G. Gordon, and S. Thrun, Point-based value iteration: An anytime algorithm for pomdps, in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, edited by G. Gottlob and T. Walsh, pp. 1025–1032, Morgan Kaufmann, Acapulco, Mexico, 2003.

[129] Pittermann, J., A. Pittermann, H. Meng, and W. Minker, Towards an emotion-sensitive spoken dialogue system - classification and dialogue modeling, in *Proceedings of the 3rd IET International Conference on Intelligent Environments (IE '07)*, pp. 239–246, Ulm, Germany, 2007.

[130] Polzin, T., and A. Waibel, Emotion-sensitive human-computer interfaces, in *Proceedings of the ISCA Tutorial and Research Workshop (ITRW) on Speech and Emotion (SpeechEmotion '00)*, pp. 201–206, Newcastle, Northern Ireland, UK, 2000.

[131] Puterman, M. L., Markov decision processes, in *Handbook in Operations Research and Management Science*, vol. 2, edited by D. Heyman and M. Sobel, pp. 331–434, Elsevier, 1990.

[132] Quarteroni, S., M. Rajman, and M. Melichar, Introducing reset patterns: an extension to a rapid dialogue prototyping methodology, in *Proceedings of the IEE International Workshop on Intelligent Environments*, pp. 276– 282, Colchester, UK, 2005.

[133] Rajman, M., R. Besanǒon, and J.-C. Chappelier, The dsir model: A distributional semantics based approach to information retrieval, *Information Retrieval-Oriented Natural Language Processing (TAL)*, *41*(2), 549–578, 2001.

[134] Rajman, M., A. Rajman, F. Seydoux, and A. Trutnev, Assessing the usability of a dialogue management system designed in the framework of a rapid dialogue prototyping methodology, in *Proceedings of the 1st ISCA Tutorial and Research Workshop on Auditory Quality of Systems (AQS-2003)*, pp. 126–133, Akademie Mont-Cenis, Germany, 2003.

[135] Rajman, M., A. Rajman, F. Seydoux, and A. Trutnev, Prototypage rapide et évaluation de modèles de dialogue finalisés, *TALN, Traitement Automatique des Langues Naturelles, Batz-sur-Mer*, 2003.

[136] Rajman, M., T. H. Bui, A. Rajman, F. Seydoux, A. Trutnev, and S. Quarteroni, Assessing the usability of a dialogue management system designed in the framework of a rapid dialogue prototyping methodology, *Acta Acustica united with Acustica, the Journal of the European Acoustics Association (EAA): International Journal on Acoustics*, *90*(6), 1096–1111, 2004.

[137] Rani, P., C. Liu, N. Sarkar, and E. Vanman, An empirical study of machine learning techniques for affect recognition in human-robot interaction, *Pattern Analysis and Applications Journal*, *9*(1), 58–69, 2006.

[138] Reeves, B., and C. Nass, *The media equation: how people treat computers, television, and new media like real people and places*, Cambridge University Press, 1996.

[139] Reilly, W. S. N., Believable social and emotional agents, Ph.D. thesis, Carnegie Mellon University, 1996.

[140] Rousseau, D., Personality in computer characters, in *Proceedings of the AAAI Workshop on Entertainment and AI/A-Life*, pp. 38–43, AAAI Press, Portland, Oregon, USA, 1996.

[141] Roy, N., and S. Thrun, Coastal navigation with mobile robots, in *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, edited by S. A. Solla, T. K. Leen, and K.-R. Müller, pp. 1043–1049, The MIT Press, Denver, Colorado, USA, 2000.

[142] Roy, N., J. Pineau, and S. Thrun, Spoken dialogue management using probabilistic reasoning, in *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-00)*, pp. 93 – 100, ACL, Hong Kong, China, 2000.

[143] Rudnicky, A., and X. Wei, An agenda-based dialog management architecture for spoken language systems, in *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU-99)*, 1999.

[144] Russell, J. A., A circumplex model of affect, *Journal of Personality and Social Psychology*, *39-6*, 1161–1178, 1980.

[145] Russell, S., and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed., 630 pp., Prentice Hall, 2003.

[146] Sacks, H., E. A. Schegloff, and G. Jefferson, A simplest systematics for the organization of turn-taking for conversation, *Language*, *50*(4), 696–735, 1974.

[147] Schatzmann, J., K. Weilhammer, M. Stuttle, and S. Young, A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies, *Knowledge Engineering Review*, *21(2)*, 97–126, 2006.

[148] Schatzmann, J., B. Thomson, K. Weilhammer, H. Ye, and S. Young, Agenda-based user simulation for bootstrapping a POMDP dialogue system, in *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT '07)*, pp. 149–152, ACL, Rochester, New York, 2007.

[149] Scheffler, K., and S. J. Young, Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning, in *Proceedings of the 2nd International Conference on Human Language Technology Research (HLT 2002)*, edited by M. Marcus, pp. 12–18, Morgan Kaufmann, 2002.

[150] Searle, J. R., *Speech Acts: An Essay in the Philosophy of Language*, 216 pp., Cambridge University Press, 1969.

[151] Sebe, N., M. S. Lew, Y. Sun, I. Cohen, T. Gevers, and T. S. Huang, Authentic facial expression analysis, *Image and Vision Computing*, *25*(12), 1856–1863, 2007.

[152] Seneff, S., R. Lau, and J. Polifroni, Organization, communication, and control in the GALAXY-II conversational system, in *Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH 99)*, pp. 1271–1274, Budapest, Hungary, 1999.

[153] Shani, G., R. I. Brafman, and S. E. Shimony, Forward search value iteration for pomdps, in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, edited by M. M. Veloso, pp. 2619–2624, Hyderabad, India, 2007.

[154] Sharma, R., M. Yeasin, N. Krahnstoever, I. Rauschert, G. Cai, I. Brewer, A. MacEachren, and K. Sengupta, Speech-gesture driven multimodal interfaces for crisis management, *Proceedings of the IEEE*, *91*(9), 28, 2003.

[155] Smallwood, R. D., and E. J. Sondik, The optimal control of partially observable Markov processes over a finite horizon, *Operations Research*, *21-5*, 1071–1088, 1973.

[156] Smith, T., and R. G. Simmons, Heuristic search value iteration for POMDPs, in *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence (UAI-04)*, edited by D. M. Chickering and J. Y. Halpern, pp. 520–527, AUAI Press, Banff, Canada, 2004.

[157] Smith, T., and R. G. Simmons, Point-based POMDP algorithms: Improved analysis and implementation, in *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence (UAI '05)*, pp. 542–547, AUAI Press, Edinburgh, Scotland, 2005.

[158] Sondik, E. J., The optimal control of partially observable markov decision processes, Ph.D. thesis, Stanford University, 1971.

[159] Song, D., Combining speech user interfaces of different applications, Ph.D. thesis, University of Munich, 2006.

[160] Spaan, M. T. J., and N. Vlassis, Perseus: randomized point-based value iteration for POMDPs, *Tech. rep.*, Universiteit van Amsterdam, 2004.

[161] Spaan, M. T. J., and N. Vlassis, Perseus: Randomized point-based value iteration for POMDPs, *Journal of Artificial Intelligence Research (JAIR)*, *24*, 195–220, 2005.

[162] Stalnaker, R. C., Assertion, *Pragmatics: Syntax and Semantics*, *9*, 315–332, 1978.

[163] Sutton, R. S., and A. G. Barto, *Reinforcement Learning: An Introduction*, 322 pp., The MIT Press, 1998.

[164] Thrun, S., W. Burgard, and D. Fox, *Probabilistic Robotics*, 667 pp., The MIT Press, 2005.

[165] Traum, D., and S. Larsson, The information state approach to dialogue management, in *Current and New Directions in Discourse and Dialogue*, edited by J. van Kuppevelt and R. W. Smith, chap. 15, pp. 325–353, Kluwer Academic Publishers, 2003.

[166] Traum, D. R., A computational theory of grounding in natural language conversation, Ph.D. thesis, University of Rochester, 1994.

[167] Traum, D. R., and E. A. Hinkelman, Conversation acts in task-oriented spoken dialogue, *Tech. Rep. TR425*, University of Rochester, 1992.

[168] Traum, D. R., and J. Rickel, Embodied agents for multi-party dialogue in immersive virtual worlds, in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pp. 766–773, ACM, Bologna, Italy, 2002.

[169] Velásquez, J. D., Modeling emotions and other motivations in synthetic agents, in *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, p. 10Ê15, AAAI Press, Providence, Rhode Island, USA, 1997.

[170] Virin, Y., G. Shani, S. E. Shimony, and R. Brafman, Scaling up: Solving POMDPs through value based clustering, in *Proceedings of the 22nd of National Conference on Artificial Intelligence (AAAI '07)*, pp. 1290–1295, AAAI Press, Vancouver, British Columbia, Canada, 2007.

[171] Vo, M. T., and C. Wood, Building an application framework for speech and pen input integration in multimodal learning interfaces, in *Proceedings of the 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-96)*, pp. 3545–3548, IEEE Computer Society, Atlanta, Georgia, USA, 1996.

[172] Vrugt, J., and T. Portele, Application-Independent Knowledge-Processing in a Task-Oriented Speech-Dialog-System, *it – Information Technology*, *46*(6), 306–314, 2004.

[173] Wahlster, W., N. Reithinger, and A. Blocher, SmartKom: Multimodal communication with a life-like character, in *Proceedings of the 7th European Conference on Speech Communication and Technology (EUROSPEECH 2001)*, vol. 3, pp. 1547–1550, Aalborg, Denmark, 2001.

[174] Williams, J., and S. Young, Scaling up POMDPs for dialogue management: the summary POMDP method., in *Proceedings of the IEEE workshop on Automatic Speech Recognition and Understanding (ASRU '05)*, pp. 250–255, Cancún, Mexico, 2005.

[175] Williams, J. D., Partially observable Markov decision processes for dialog management, Ph.D. thesis, Cambridge University, 2006.

[176] Williams, J. D., and S. Young, Scaling POMDPs for dialog management with composite summary point-based value iteration (CSPBVI), in *Proceedings of the AAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems*, pp. 37–42, The AAAI Press, Boston, MA, USA, 2006.

[177] Williams, J. D., and S. Young, Partially observable markov decision processes for spoken dialog systems, *Computer Speech and Language*, *21*(2), 393–422, 2007.

[178] Williams, J. D., P. Poupart, and S. Young, Partially observable Markov decision processes with continuous observations for dialogue management, in *Proceedings of the 6th SigDial Workshop on Discourse and Dialogue (SIGdial '05)*, 2005.

[179] Williams, J. D., P. Poupart, and S. Young, Factored partially observable Markov decision processes for dialogue management, in *Proceedings of the 4th Workshop on Knowledge and Reasoning in Practical Dialog Systems (KRPD '05)*, edited by I. Zukerman, J. Alexandersson, and A. Jönsson, pp. 76–82, Edinburgh, Scotland, 2005.

[180] Wu, L., S. L. Oviatt, and P. R. Cohen, From members to teams to committee - a robust approach to gestural and multimodal recognition, *IEEE Transactions on Neural Networks*, *13*(4), 11, 2002.

[181] Yacoub, S., S. Simske, X. Lin, and J. Burns, Recognition of emotions in interactive voice response systems, in *Proceedings of the 8th European Conference on Speech Communication and Technology (EUROSPEECH 2003)*, pp. 729–732, Geneva, Switzerland, 2003.

[182] Young, S., Talking to machines (statistically speaking), in *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP 2002)*, pp. 9–16, Denver, Colorado, USA, 2002.

[183] Young, S., J. D. Williams, J. Schatzmann, M. Stuttle, and K. Weilhammer, The hidden information state approach to dialogue management, *Tech. Rep. CUED/F-INFENG/TR.544*, University of Cambridge, 2005.

[184] Young, S., J. Schatzmann, B. Thomson, K. Weilhammer, and H. Ye, The hidden information state dialogue manager: A real-world pomdp-based system, in *Proceedings of the NAACL HLT Demonstration Program (NAACL-HLT '07)*, pp. 27–28, ACL, Rochester, New York, USA, 2007.

[185] Young, S., J. Schatzmann, K. Weilhammer, and H. Ye, The hidden information state approach to dialogue management, in *Proceedings of the 2007 International Conference on Acoustics, Speech, and Signal Processing (ICASSP '07)*, vol. 4, pp. 149–152, Honolulu, Hawaii, USA, 2007.

[186] Zeng, Z., M. Pantic, G. I. Roisman, and T. S. Huang, A survey of affect recognition methods: audio, visual and spontaneous expressions, in *Proceedings of the 9th International Conference on Multimodal interfaces (ICMI '07)*, edited by D. W. Massaro, K. Takeda, D. Roy, and A. Potamianos, pp. 126–133, ACM, Nagoya, Aichi, Japan, 2007.

[187] Zhang, B., Q. Cai, J. Mao, and B. Guo, Spoken dialog management as planning and acting under uncertainty, in *Proceedings of the 7th European Conference on Speech Communication and Technology (EUROSPEECH '01)*, pp. 2169–2172, Aalborg, Denmark, 2001.

[188] Zhang, N. L., Efficient planning in stochastic domains through exploiting problem characteristics, *Tech. Rep. HKUST-CS95-40*, Hong Kong University of Science and Technology, 1995.

[189] Zhang, N. L., and W. Liu, Planning in stochastic domains: Problem characteristics and approximation, *Tech. Rep. HKUST-CS96-31*, Hong Kong Univesity of Science and Technology, 1996.

[190] Zhang, N. L., and W. Zhang, Speeding up the convergence of value iteration in partially observable markov decision processes, *Journal of Artificial Intelligence Reseasrch (JAIR)*, *14*, 29–51, 2001.

[191] Zhou, X., and C. Conati, Inferring user goals from personality and behavior in a causal model of user affect, in *Proceedings of the 8th international conference on Intelligent user interfaces (IUI '03)*, pp. 211–218, ACM, Miami, FL, USA, 2003.

Abstract

Designing and developing affective dialogue systems have recently received much interest from the dialogue research community. A distinctive feature of these systems is affect modeling. Previous work was mainly focused on showing system's emotions to the user in order to achieve the designer's goal such as helping the student to practice nursing tasks or persuading the user to change their dietary behavior. A challenging problem is to infer the user's affective state and to adapt the system's behavior accordingly. This thesis addresses this problem from an engineering perspective using Partially Observable Markov Decision Process (POMDP) techniques and a Rapid Dialogue Prototyping Methodology (RDPM).

We argue that the POMDPs are suitable for use in designing affective dialogue management models for three main reasons. First, the POMDP model allows for realistic modeling of the user's affective state, the user's intention, and other (user's) hidden state components by incorporating them into the state space. Second, recent dialogue management research has shown that the POMDP-based dialogue manager is able to cope well with uncertainty that can occur at many levels inside a dialogue system from speech recognition, natural language understanding to dialogue management. Third, the POMDP environment can be used to create a simulated user which is useful for learning and evaluation of competing dialogue strategies.

In the first part of this thesis, we first present the RDPM for a quick production of frame-based dialogue models for traditional (i.e., non-affect sensitive) single-application dialogue systems. The usability of the RDPM has been validated through the implementation of several prototype dialogue systems. We then present a novel approach to developing interfaces for multi-application systems which are dialogue systems that allow the user to navigate between a large set of applications smoothly and transparently. The work in this part provides an essential infrastructure for implementing our prototype POMDP-based dialogue manager.

In the second part, we first describe a factored POMDP approach to affective dialogue management. This approach illustrates that POMDPs are an elegant model for building affective dialogue systems. Further, the POMDP-based dialogue strategy outperforms all other known strategies from the literature when tested with small-scale dialogue problems. However, a well-known drawback of POMDP-based dialogue managers is that computing a near-optimal dialogue policy is extremely computationally expensive. We then propose a tractable hybrid DDN-POMDP method to tackle many of these scalability problems. The central contribution of our method (compared with other POMDP-based dialogue management methods from the literature) is the ability to handle frame-based dialogue problems with hundreds of slots and hundreds of slot values.

Keywords: dialogue modeling, dialogue management, dialogue systems, rapid prototyping, partially observable Markov decision processes, multimodal, multi-application, multi-domain, affective computing.

Samenvatting

Het ontwerpen en ontwikkelen van affectieve dialoogsystemen heeft de afgelopen tijd veel belangstelling genoten binnen het onderzoek naar dialogen. Een bijzonder kenmerk van deze systemen is het modelleren van emoties. Het voorafgaande werk was voornamelijk gericht op het weergeven van de emoties van het systeem naar de gebruiker om het doel van de ontwerper te bereiken, zoals het helpen van een student bij het oefenen van taken in de verpleging, of het helpen van de gebruiker zijn of haar eetgedrag aan te passen ten behoeve van een dieet. Een uitdagend probleem is het afleiden van de gemoedstoestand van de gebruiker en het gedrag van het systeem daarop aan te sluiten. Dit proefschrift gaat op dit probleem in vanuit een praktisch perspectief met behulp van de technieken Partially Observable Markov Decision Process (POMDP) en Rapid Dialogue Prototyping Methodology (RDPM).

Wij tonen aan dat POMDP's geschikt zijn om te gebruiken bij het ontwerpen van affectieve modellen van dialoogmanagement en wel om drie redenen. Ten eerste maakt het POMDP-model een realistische modellering mogelijk van de gemoedstoestand van de gebruiker, de bedoeling van de gebruiker, en andere componenten van een verborgen toestand (van de gebruiker), door ze in te bedden in de toestandsruimte. Ten tweede heeft recent onderzoek naar dialoogmanagement aangetoond dat dialoogmanagers gebaseerd op POMDP's, goed kunnen omgaan met onzekerheden die kunnen optreden op allerlei niveaus binnen een dialoogsysteem, van spraakherkenning, natuurlijke taalverwerking tot dialoogmanagement. Ten derde kan de POMDP-omgeving worden gebruikt om een gesimuleerde gebruiker te maken, wat nuttig is voor het leren en evalueren van alternatieve dialoogstrategieọn.

In het eerste deel van dit proefschrift zullen we eerst de RDPM voorstellen voor het snel produceren van op frames gebaseerde dialoogmodellen voor traditionele (d.w.z. niet-affectieve) dialoogsystemen voor ȭȭn toepassing. De bruikbaarheid van de RDPM is gevalideerd door de implementatie van verschillende prototypen van dialoogsystemen. Daarna stellen we een nieuwe benadering voor om interfaces te ontwikkelen voor systemen voor meerdere toepassingen. Dit zijn dialoogsystemen waarin de gebruiker transparant en zonder merkbare overgangen kan navigeren binnen een grote verzameling van toepassingen. Het werk in dit deel vormt een essentiọle infrastructuur voor het implementeren van ons prototype van een dialoogmanager gebaseerd op POMDP's.

In het tweede deel beschrijven we eerst een een benadering van gefactoreerde POMDP's voor affectief dialoogmanagement. Deze benadering illustreert dat POMDP's

een elegant model zijn voor het bouwen van affectieve dialoogsystemen. Bovendien overtreft de prestatie van de POMDP-gebaseerde dialoogstrategie alle overige bekende strategieën uit de literatuur, zodra ze getest worden met kleinschalige dialoogproblemen. Een bekende tekortkoming van POMDP-gebaseerde dialoogmanagers is echter dat het berekenen van een bijna optimaal dialoogbeleid buitengewoon veel rekenkracht vereist. We stellen dan een haalbare hybride methode van DDN-POMDP voor om veel van deze schaalbaarheidsproblemen het hoofd te bieden. De voornaamste bijdrage van onze methode (vergeleken met andere POMDP-gebaseerde methoden van dialoogmanagement uit de literatuur) is de mogelijkheid op frames gebaseerde dialoogproblemen te verwerken met honderden slots en honderden slotwaarden.

Zoekwoorden: dialoogmodellering, dialoogmanagement, dialoogsystemen, rapid prototyping, partially observable Markov decision processes, multimodaal, meerdere toepassingen, meerdere domeinen, affectieve computers

Curriculum Vitae

Trung H. Bui (Vietnamese: Bùi Hữu Trung) was born on July 5, 1975, in, Vinh, a city located in the north central coast of Vietnam. He attended the special school for mathematically gifted pupils at the Vinh University and was selected two times as a member of the university team to attend the National Mathematics Competitions in 1991 and 1992, respectively. In 1992, at the age of 17, he moved to Hanoi to study computer science at the Hanoi University of Technology. In 1997, after graduating with an engineer's degree, he worked for several ICT companies as a software and network engineer and finished a Master program in computer science at the Francophone Institute for Informatics. In 2002, he went to work at the Artificial Intelligence Laboratory, Swiss Federal Institute of Technology in Lausanne (EPFL), first as a research intern and then as a research assistant. He also took a one-year pre-doctoral program at the School of Computer and Communication Sciences, EPFL. In November 2004, he moved to Enschede to work for the Human Media Interaction (HMI) group, University of Twente, as as a research assistant and a PhD student. The main part of this thesis describes the results of his work carried out at the HMI group.

SIKS Dissertation Series

Since 1998, all dissertations written by Ph.D. students who have conducted their research under auspices of a senior research fellow of the SIKS research school are published in the SIKS Dissertation Series. This thesis is the 196th in the series.

2008-32 Trung H. Bui (UT), *Toward Affective Dialogue Management Using Partially Observable Markov Decision Processes*
2008-31 Loes Braun (UM), *Pro-Active Medical Information Retrieval*
2008-30 Wouter van Atteveldt (VU), *Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content*
2008-29 Dennis Reidsma (UT), *Annotations and Subjective Machines – Of Annotators, Embodied Agents, Users, and Other Humans*
2008-28 Ildiko Flesch (RUN), *On the Use of Independence Relations in Bayesian Networks*
2008-27 Hubert Vogten (OU), *Design and Implementation Strategies for IMS Learning Design*
2008-26 Marijn Huijbregts (UT), *Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled*
2008-25 Geert Jonker (UU), *Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency*
2008-24 Zharko Aleksovski (VU), *Using background knowledge in ontology matching*
2008-23 Stefan Visscher (UU), *Bayesian network models for the management of ventilator-associated pneumonia*
2008-22 Henk Koning (UU), *Communication of IT-Architecture*
2008-21 Krisztian Balog (UVA), *People Search in the Enterprise*
2008-20 Rex Arendsen (UVA), *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.*
2008-19 Henning Rode (UT), *From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search*
2008-18 Guido de Croon (UM), *Adaptive Active Vision*
2008-17 Martin Op 't Land (TUD), *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*
2008-16 Henriëtte van Vugt (VU), *Embodied agents from a user's perspective*
2008-15 Martijn van Otterlo (UT), *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.*
2008-14 Arthur van Bunningen (UT), *Context-Aware Querying; Better Answers with Less Effort*
2008-13 Caterina Carraciolo (UVA), *Topic Driven Access to Scientific Handbooks*
2008-12 József Farkas (RUN), *A Semiotically Oriented Cognitive Model of Knowledge Representation*
2008-11 Vera Kartseva (VU), *Designing Controls for Network Organizations: A Value-Based Approach*
2008-10 Wauter Bosma (UT), *Discourse oriented summarization*
2008-09 Christof van Nimwegen (UU), *The paradox of the guided user: assistance can be counter-effective*
2008-08 Janneke Bolt (UU), *Bayesian Networks: Aspects of Approximate Inference*
2008-07 Peter van Rosmalen (OU), *Supporting the tutor in the design and support of adaptive e-learning*
2008-06 Arjen Hommersom (RUN), *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective*
2008-05 Bela Mutschler (UT), *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*
2008-04 Ander de Keijzer (UT), *Management of Uncertain Data – towards unattended integration*
2008-03 Vera Hollink (UVA), *Optimizing hierarchical menus: a usage-based approach*
2008-02 Alexei Sharpanskykh (VU), *On Computer-Aided Methods for Modeling and Analysis of Organizations*
2008-01 Katalin Boer-Sorbán (EUR), *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*
2007-25 Joost Schalken (VU), *Empirical Investigations in Software Process Improvement*
2007-24 Georgina Ramírez Camps (CWI), *Structural Features in XML Retrieval*
2007-23 Peter Barna (TUE), *Specification of Application Logic in Web Information Systems*

2007-22 Zlatko Zlatev (UT), *Goal-oriented design of value and process models from patterns*
2007-21 Karianne Vermaas (UU), *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*
2007-20 Slinger Jansen (UU), *Customer Configuration Updating in a Software Supply Network*
2007-19 David Levy (UM), *Intimate relationships with artificial partners*
2007-18 Bart Orriëns (UvT), *On the development an management of adaptive business collaborations*
2007-17 Theodore Charitos (UU), *Reasoning with Dynamic Networks in Practice*
2007-16 Davide Grossi (UU), *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems*
2007-15 Joyca Lacroix (UM), *NIM: a Situated Computational Memory Model*
2007-14 Niek Bergboer (UM), *Context-Based Image Analysis*
2007-13 Rutger Rienks (UT), *Meetings in Smart Environments; Implications of Progressing Technology*
2007-12 Marcel van Gerven (RUN), *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*
2007-11 Natalia Stash (TUE), *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*
2007-10 Huib Aldewereld (UU), *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*
2007-09 David Mobach (VU), *Agent-Based Mediated Service Negotiation*
2007-08 Mark Hoogendoorn (VU), *Modeling of Change in Multi-Agent Organizations*
2007-07 Nataša Jovanović (UT), *To Whom It May Concern – Addressee Identification in Face-to-Face Meetings*
2007-06 Gilad Mishne (UVA), *Applied Text Analytics for Blogs*
2007-05 Bart Schermer (UL), *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*
2007-04 Jurriaan van Diggelen (UU), *Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach*
2007-03 Peter Mika (VU), *Social Networks and the Semantic Web*
2007-02 Wouter Teepe (RUG), *Reconciling Information Exchange and Confidentiality: A Formal Approach*
2007-01 Kees Leune (UvT), *Access Control and Service-Oriented Architectures*
2006-28 Börkur Sigurbjörnsson (UVA), *Focused Information Access using XML Element Retrieval*
2006-27 Stefano Bocconi (CWI), *Vox Populi: generating video documentaries from semantically annotated media repositories*
2006-26 Vojkan Mihajlović (UT), *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*
2006-25 Madalina Drugan (UU), *Conditional log-likelihood MDL and Evolutionary MCMC*
2006-24 Laura Hollink (VU), *Semantic Annotation for Retrieval of Visual Resources*
2006-23 Ion Juvina (UU), *Development of Cognitive Model for Navigating on the Web*
2006-22 Paul de Vrieze (RUN), *Fundaments of Adaptive Personalisation*
2006-21 Bas van Gils (RUN), *Aptness on the Web*
2006-20 Marina Velikova (UvT), *Monotone models for prediction in data mining*
2006-19 Birna van Riemsdijk (UU), *Cognitive Agent Programming: A Semantic Approach*
2006-18 Valentin Zhizhkun (UVA), *Graph transformation for Natural Language Processing*
2006-17 Stacey Nagata (UU), *User Assistance for Multitasking with Interruptions on a Mobile Device*
2006-16 Carsten Riggelsen (UU), *Approximation Methods for Efficient Learning of Bayesian Networks*
2006-15 Rainer Malik (UU), *CONAN: Text Mining in the Biomedical Domain*
2006-14 Johan Hoorn (VU), *Software Requirements: Update, Upgrade, Redesign – towards a Theory of Requirements Change*
2006-13 Henk-Jan Lebbink (UU), *Dialogue and Decision Games for Information Exchanging Agents*
2006-12 Bert Bongers (VU), *Interactivation – Towards an e-cology of people, our technological environment, and the arts*
2006-11 Joeri van Ruth (UT), *Flattening Queries over Nested Data Types*
2006-10 Ronny Siebes (VU), *Semantic Routing in Peer-to-Peer Systems*
2006-09 Mohamed Wahdan (UM), *Automatic Formulation of the Auditor's Opinion*
2006-08 Eelco Herder (UT), *Forward, Back and Home Again – Analyzing User Behavior on the Web*
2006-07 Marko Smiljanic (UT), *XML schema matching – balancing efficiency and effectiveness by means of clustering*
2006-06 Ziv Baida (VU), *Software-aided Service Bundling – Intelligent Methods & Tools for Graphical Service Modeling*
2006-05 Cees Pierik (UU), *Validation Techniques for Object-Oriented Proof Outlines*
2006-04 Marta Sabou (VU), *Building Web Service Ontologies*
2006-03 Noor Christoph (UVA), *The role of metacognitive skills in learning to solve problems*
2006-02 Cristina Chisalita (VU), *Contextual issues in the design and use of information technology in organizations*
2006-01 Samuil Angelov (TUE), *Foundations of B2B Electronic Contracting*
2005-21 Wijnand Derks (UT), *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*
2005-20 Cristina Coteanu (UL), *Cyber Consumer Law, State of the Art and Perspectives*
2005-19 Michel van Dartel (UM), *Situated Representation*
2005-18 Danielle Sent (UU), *Test-selection strategies for probabilistic networks*
2005-17 Boris Shishkov (TUD), *Software Specification Based on Re-usable Business Components*

2005-16 Joris Graaumans (UU), *Usability of XML Query Languages*
2005-15 Tibor Bosse (VU), *Analysis of the Dynamics of Cognitive Processes*
2005-14 Borys Omelayenko (VU), *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*
2005-13 Fred Hamburg (UL), *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*
2005-12 Csaba Boer (EUR), *Distributed Simulation in Industry*
2005-11 Elth Ogston (VU), *Agent Based Matchmaking and Clustering – A Decentralized Approach to Search*
2005-10 Anders Bouwer (UVA), *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*
2005-09 Jeen Broekstra (VU), *Storage, Querying and Inferencing for Semantic Web Languages*
2005-08 Richard Vdovjak (TUE), *A Model-driven Approach for Building Distributed Ontology-based Web Applications*
2005-07 Flavius Frasincar (TUE), *Hypermedia Presentation Generation for Semantic Web Information Systems*
2005-06 Pieter Spronck (UM), *Adaptive Game AI*
2005-05 Gabriel Infante-Lopez (UVA), *Two-Level Probabilistic Grammars for Natural Language Parsing*
2005-04 Nirvana Meratnia (UT), *Towards Database Support for Moving Object data*
2005-03 Franc Grootjen (RUN), *A Pragmatic Approach to the Conceptualisation of Language*
2005-02 Erik van der Werf (UM)), *AI techniques for the game of Go*
2005-01 Floor Verdenius (UVA), *Methodological Aspects of Designing Induction-Based Applications*
2004-20 Madelon Evers (Nyenrode), *Learning from Design: facilitating multidisciplinary design teams*
2004-19 Thijs Westerveld (UT), *Using generative probabilistic models for multimedia retrieval*
2004-18 Vania Bessa Machado (UvA), *Supporting the Construction of Qualitative Knowledge Models*
2004-17 Mark Winands (UM), *Informed Search in Complex Games*
2004-16 Federico Divina (VU), *Hybrid Genetic Relational Search for Inductive Learning*
2004-15 Arno Knobbe (UU), *Multi-Relational Data Mining*
2004-14 Paul Harrenstein (UU), *Logic in Conflict. Logical Explorations in Strategic Equilibrium*
2004-13 Wojciech Jamroga (UT), *Using Multiple Models of Reality: On Agents who Know how to Play*
2004-12 The Duy Bui (UT), *Creating emotions and facial expressions for embodied agents*
2004-11 Michel Klein (VU), *Change Management for Distributed Ontologies*
2004-10 Suzanne Kabel (UVA), *Knowledge-rich indexing of learning-objects*
2004-09 Martin Caminada (VU), *For the Sake of the Argument; explorations into argument-based reasoning*
2004-08 Joop Verbeek (UM), *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiÂle gegevensuitwisseling en digitale expertise*
2004-07 Elise Boltjes (UM), *Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes*
2004-06 Bart-Jan Hommes (TUD), *The Evaluation of Business Process Modeling Techniques*
2004-05 Viara Popova (EUR), *Knowledge discovery and monotonicity*
2004-04 Chris van Aart (UVA), *Organizational Principles for Multi-Agent Architectures*
2004-03 Perry Groot (VU), *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*
2004-02 Lai Xu (UvT), *Monitoring Multi-party Contracts for E-business*
2004-01 Virginia Dignum (UU), *A Model for Organizational Interaction: Based on Agents, Founded in Logic*
2003-18 Levente Kocsis (UM), *Learning Search Decisions*
2003-17 David Jansen (UT), *Extensions of Statecharts with Probability, Time, and Stochastic Timing*
2003-16 Menzo Windhouwer (CWI), *Feature Grammar Systems – Incremental Maintenance of Indexes to Digital Media Warehouses*
2003-15 Mathijs de Weerdt (TUD), *Plan Merging in Multi-Agent Systems*
2003-14 Stijn Hoppenbrouwers (KUN), *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*
2003-13 Jeroen Donkers (UM), *Nosce Hostem – Searching with Opponent Models*
2003-12 Roeland Ordelman (UT), *Dutch speech recognition in multimedia information retrieval*
2003-11 Simon Keizer (UT), *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*
2003-10 Andreas Lincke (UvT), *Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture*
2003-09 Rens Kortmann (UM), *The resolution of visually guided behaviour*
2003-08 Yongping Ran (UM), *Repair Based Scheduling*
2003-07 Machiel Jansen (UvA), *Formal Explorations of Knowledge Intensive Tasks*
2003-06 Boris van Schooten (UT), *Development and specification of virtual environments*
2003-05 Jos Lehmann (UVA), *Causation in Artificial Intelligence and Law – A modelling approach*
2003-04 Milan Petković (UT), *Content-Based Video Retrieval Supported by Database Technology*
2003-03 Martijn Schuemie (TUD), *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*
2003-02 Jan Broersen (VU), *Modal Action Logics for Reasoning About Reactive Systems*
2003-01 Heiner Stuckenschmidt (VU), *Ontology-Based Information Sharing in Weakly Structured Environments*

2002-17 Stefan Manegold (UVA), *Understanding, Modeling, and Improving Main-Memory Database Performance*
2002-16 Pieter van Langen (VU), *The Anatomy of Design: Foundations, Models and Applications*
2002-15 Rik Eshuis (UT), *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*
2002-14 Wieke de Vries (UU), *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*
2002-13 Hongjing Wu (TUE), *A Reference Architecture for Adaptive Hypermedia Applications*
2002-12 Albrecht Schmidt (Uva), *Processing XML in Database Systems*
2002-11 Wouter C.A. Wijngaards (VU), *Agent Based Modelling of Dynamics: Biological and Organisational Applications*
2002-10 Brian Sheppard (UM), *Towards Perfect Play of Scrabble*
2002-09 Willem-Jan van den Heuvel (KUB), *Integrating Modern Business Applications with Objectified Legacy Systems*
2002-08 Jaap Gordijn (VU), *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*
2002-07 Peter Boncz (CWI), *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*
2002-06 Laurens Mommers (UL), *Applied legal epistemology; Building a knowledge-based ontology of the legal domain*
2002-05 Radu Serban (VU), *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents*
2002-04 Juan Roberto Castelo Valdueza (UU), *The Discrete Acyclic Digraph Markov Model in Data Mining*
2002-03 Henk Ernst Blok (UT), *Database Optimization Aspects for Information Retrieval*
2002-02 Roelof van Zwol (UT), *Modelling and searching web-based document collections*
2002-01 Nico Lassing (VU), *Architecture-Level Modifiability Analysis*
2001-11 Tom M. van Engers (VUA), *Knowledge Management: The Role of Mental Models in Business Systems Design*
2001-10 Maarten Sierhuis (UvA), *Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design*
2001-09 Pieter Jan 't Hoen (RUL), *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*
2001-08 Pascal van Eck (VU), *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*
2001-07 Bastiaan Schonhage (VU), *Diva: Architectural Perspectives on Information Visualization*
2001-06 Martijn van Welie (VU), *Task-based User Interface Design*
2001-05 Jacco van Ossenbruggen (VU), *Processing Structured Hypermedia: A Matter of Style*
2001-04 Evgueni Smirnov (UM), *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*
2001-03 Maarten van Someren (UvA), *Learning as problem solving*
2001-02 Koen Hindriks (UU), *Agent Programming Languages: Programming with Mental Models*
2001-01 Silja Renooij (UU), *Qualitative Approaches to Quantifying Probabilistic Networks*
2000-11 Jonas Karlsson (CWI), *Scalable Distributed Data Structures for Database Management*
2000-10 Niels Nes (CWI), *Image Database Management System Design Considerations, Algorithms and Architecture*
2000-09 Florian Waas (CWI), *Principles of Probabilistic Query Optimization*
2000-08 Veerle Coupé (EUR), *Sensitivity Analyis of Decision-Theoretic Networks*
2000-07 Niels Peek (UU), *Decision-theoretic Planning of Clinical Patient Management*
2000-06 Rogier van Eijk (UU), *Programming Languages for Agent Communication*
2000-05 Ruud van der Pol (UM), *Knowledge-based Query Formulation in Information Retrieval.*
2000-04 Geert de Haan (VU), *ETAG, A Formal Model of Competence Knowledge for User Interface Design*
2000-03 Carolien M.T. Metselaar (UVA), *Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.*
2000-02 Koen Holtman (TUE), *Prototyping of CMS Storage Management*
2000-01 Frank Niessink (VU), *Perspectives on Improving Software Maintenance*
1999-08 Jacques H.J. Lenting (UM), *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.*
1999-07 David Spelt (UT), *Verification support for object database design*
1999-06 Niek J.E. Wijngaards (VU), *Re-design of compositional systems*
1999-05 Aldo de Moor (KUB), *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*
1999-04 Jacques Penders (UM), *The practical Art of Moving Physical Objects*
1999-03 Don Beal (UM), *The Nature of Minimax Search*
1999-02 Rob Potharst (EUR), *Classification using decision trees and neural nets*
1999-01 Mark Sloof (VU), *Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products*
1998-05 E.W. Oskamp (RUL), *Computerondersteuning bij Straftoemeting*
1998-04 Dennis Breuker (UM), *Memory versus Search in Games*
1998-03 Ans Steuten (TUD), *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*
1998-02 Floris Wiesman (UM), *Information Retrieval by Graphically Browsing Meta-Information*
1998-01 Johan van den Akker (CWI), *DEGAS – An Active, Temporal Database of Autonomous Objects*