Theory and Applications of Natural Language Processing
Monographs

Verena Rieser
Oliver Lemon

Reinforcement Learning for Adaptive Dialogue Systems

A Data-driven Methodology for Dialogue Management and Natural Language Generation

Springer

Theory and Applications
of Natural Language Processing

Series Editors:
Graeme Hirst (Textbooks)
Eduard Hovy (Edited volumes)
Mark Johnson (Monographs)

Aims and Scope

The field of Natural Language Processing (NLP) has expanded explosively over the past decade: growing bodies of available data, novel fields of applications, emerging areas and new connections to neighboring fields have all led to increasing output and to diversification of research.

"Theory and Applications of Natural Language Processing" is a series of volumes dedicated to selected topics in NLP and Language Technology. It focuses on the most recent advances in all areas of the computational modeling and processing of speech and text across languages and domains. Due to the rapid pace of development, the diversity of approaches and application scenarios are scattered in an ever-growing mass of conference proceedings, making entry into the field difficult for both students and potential users. Volumes in the series facilitate this first step and can be used as a teaching aid, advanced-level information resource or a point of reference.

The series encourages the submission of research monographs, contributed volumes and surveys, lecture notes and textbooks covering research frontiers on all relevant topics, offering a platform for the rapid publication of cutting-edge research as well as for comprehensive monographs that cover the full range of research on specific problem areas.

The topics include applications of NLP techniques to gain insights into the use and functioning of language, as well as the use of language technology in applications that enable communication, knowledge management and discovery such as natural language generation, information retrieval, question-answering, machine translation, localization and related fields.

The books are available in printed and electronic (e-book) form:

* Downloadable on your PC, e-reader or iPad
* Enhanced by Electronic Supplementary Material, such as algorithms, demonstrations, software, images and videos
* Available online within an extensive network of academic and corporate R&D libraries worldwide
* Never out of print thanks to innovative print-on-demand services
* Competitively priced print editions for eBook customers thanks to MyCopy service http://www.springer.com/librarians/e-content/mycopy

For further volumes:
http://www.springer.com/series/8899

Verena Rieser • Oliver Lemon

Reinforcement Learning for Adaptive Dialogue Systems

A Data-driven Methodology for Dialogue Management and Natural Language Generation

Springer

Verena Rieser
School of Mathematical
and Computer Sciences
Heriot-Watt University
Edinburgh EH14 4AS
United Kingdom
v.t.rieser@hw.ac.uk

Oliver Lemon
School of Mathematical
and Computer Sciences
Heriot-Watt University
Edinburgh EH14 4AS
United Kingdom
o.lemon@hw.ac.uk

ISSN 2192-032X e-ISSN 2192-0338
ISBN 978-3-642-24941-9 e-ISBN 978-3-642-24942-6
DOI 10.1007/978-3-642-24942-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011942326

Mathematics Subject Classification (2010): 68-XX, 68Txx, 68T05, 68T50, 68T37, 68T42

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The past decade has seen a revolution in the field of spoken dialogue systems. As in other areas of Computer Science and Artificial Intelligence, data-driven methods are now being used to drive new methodologies for system development and evaluation. These methods are proving to be more robust, flexible, and adaptive than the largely rule-based approaches which preceded them.

We hope that this book is a contribution to that ongoing change. It describes, in detail, a new methodology for developing spoken dialogue systems – in particular the Dialogue Management and Natural Language Generation components – which starts with human data, and culminates in evaluation with real users. The journey therefore starts and ends with human behaviour in interaction, and explores methods for learning from the data, for building simulation environments for training and testing systems, and for evaluating the results.

The detailed material covers: Spoken and Multimodal dialogue systems, Wizard-of-Oz data collection, User Simulation methods, Reinforcement Learning, and Evaluation methodologies.

This book is therefore intended as research guide which navigates through a detailed case study in data-driven methods for development and evaluation of spoken dialogue systems. Common challenges associated with this approach are discussed and example solutions provided, for example, how to learn from limited amounts of data. As such, we hope it will provide insights, lessons, and inspiration for future research and development – not only for spoken dialogue systems in particular, but for data-driven approaches to human-machine interaction in general.

Edinburgh, *Verena Rieser*
September 2011 *Oliver Lemon*

Acknowledgements

The research presented here would not have been possible without interaction with our research community, and we owe special thanks to the participants of the TALK and CLASSiC projects for discussing the ideas presented in this book.

There is a lively international research community in Spoken Dialogue Systems whose interest, criticism, and encouragement have helped to shape this work. We are especially indebted to Steve Young and his colleagues at Cambridge University, who have had a deep influence on our ideas. Members of the Interaction Lab[1] at Heriot-Watt University's School of Mathematics and Computer Science have also helped to discuss and develop these ideas over several years.

In particular we would also like to thank Dr. Xingkun Liu, Dr. Helen Hastie, Dr. Ivana Kruijff-Korbayová, Dr. Tilman Becker, and other colleagues from Saarland and Edinburgh Universities for helping with the data collection and evaluation involved in this work. We especially thank Professor Manfred Pinkal for his guidance.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7, 2007-2013) under grant agreement number 216594 ("Computational Learning in Adaptive Systems for Spoken Conversation", CLASSiC project: www.classic-project.org), the EC FP6 project "TALK: Talk and Look, Tools for Ambient Linguistic Knowledge" (IST 507802, www.talk-project.org), from EPSRC grant numbers EP/E019501/1 and EP/G069840/1, and from the International Research Training Group "Language Technology and Cognitive Systems", Saarland University, funded by the German Science Foundation (DFG).

Finally, the first author would like to thank her parents Franz and Tatjana Rieser for their support and encouragement. The second author thanks his family for the decades of rewarding learning experiences which have made this book possible.

[1] http://www.macs.hw.ac.uk/InteractionLab/

Contents

Acronyms

ASR	Automatic Speech Recognition
DA	Dialogue Act
DB	Database
DM	Dialogue Management
GUI	Graphical User Interface
HCI	Human Computer Interaction
IP	Information Presentation
ISU	Information State Update
MDP	Markov Decision Process
ML	Machine Learning
NLG	Natural Language Generation
NLP	Natural Language Processing
PARADISE	PARAdigm for DIalogue System Evaluation
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
SA	Speech Act
SASSI	Subjective Assessment of Speech System Interfaces
SDS	Spoken Dialogue System
SL	Supervised Learning
SLU	Spoken Language Understanding
TTS	Text-to-Speech
VOIP	Voice-Over-Internet Protocol
WER	Word-Error Rate
WOZ	Wizard-of-Oz

Chapter 1
Introduction

The past decade has seen something of a revolution in the field of spoken dialogue systems. As in other areas of Computer Science and Artificial Intelligence, data-driven methods are being used to drive new methodologies for system development and evaluation. These methods are proving to be more robust, flexible, and adaptive than the rule-based approaches which preceded them.

We hope that this book makes a contribution to that revolution. It describes, in detail, a new methodology for developing spoken dialogue systems – in particular the Dialogue Management component – which starts with human data, and ends with evaluation with real users. Related methods are now being developed further by a number of researchers worldwide. The journey begins and ends with human behaviour in interaction, and en route we explore methods for learning from such data, for building simulation environments for training and testing our systems, and methods for evaluating the results.

This book is therefore intended as a guide which navigates through a detailed case study in data-driven methods for development and evaluation of spoken dialogue systems. It focusses on Dialogue Management and Natural Language Generation, rather than speech recognition and spoken language understanding. As such, we hope that it can provide insights and lessons for future research and development – not only for spoken dialogue systems in particular, but for data-driven approaches to building better human-machine interaction in general.

1.1 The Design Problem for Spoken Dialogue Systems

The design of Spoken Dialogue Systems (SDS) is not only concerned with integrating speech and language processing modules such as Automatic Speech Recognition (ASR), Spoken Language Understanding (SLU), Natural Language Generation (NLG), and Text-to-Speech (TTS) synthesis systems. It also requires the development of skills for "what to say next": *dialogue strategies* which take into account the performance of these components, the nature of the user's tasks (e.g.

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*, Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_1,

information-seeking, tutoring, or robot control), and other features of the operating environment such as the user's behaviour and preferences. The great variability and unpredictability of these factors makes dialogue strategy design an extremely difficult task for human developers. In conventional, rule-based, dialogue development many expensive iterations of manual design and re-design are necessary in order to produce good strategies. In addition, such hand-coded strategies are not re-usable from task to task, are not scalable, require a substantial amount of human labour and expertise, and are not guaranteed to be optimal.

For these reasons machine learning methods (such as Reinforcement Learning) for dialogue strategy design have been a leading research area for several years. These statistical computational learning approaches offer several key potential advantages over the standard rule-based hand-coding approach to dialogue systems development (Lemon and Pietquin, 2007):

- a data-driven automatic development cycle
- provably optimal action policies
- a principled mathematical model for action selection
- possibilities for generalisation to unseen states
- reduced development and deployment costs.

However, in cases where a system is designed from scratch, there is often no suitable in-domain data to enable such a design. Collecting dialogue data without a working prototype is problematic, leaving the developer with a classic "chicken-and-egg" problem. One of the main issues that this book addresses is how to use a data-driven development methodology when little or no in-domain data exists.

1.2 Overview

In this book we propose to learn dialogue strategies by simulation-based Reinforcement Learning (RL) (Sutton and Barto, 1998), where a simulated environment is learned from small amounts of Wizard-of-Oz (WOZ) data. Using WOZ data rather than data from real Human-Computer Interaction (HCI) allows us to learn optimal strategies for domains where no working dialogue system already exists. Automatic strategy learning has been applied to dialogue systems which have already been deployed in the real world using handcrafted strategies. In such work, strategy learning was performed based on already present extensive online-operation experience, e.g. (Henderson et al, 2005, 2008; Singh et al, 2002). In contrast to this preceding work, our approach enables strategy learning in domains where no prior system is available. Optimised learned strategies are then available from the first moment of online-operation, and labour-intensive handcrafting of dialogue strategies is avoided. This independence from large amounts of in-domain dialogue data allows researchers to apply RL to new application areas beyond the scope of existing dialogue systems. We call this method "bootstrapping".

This book first provides the general proof-of-concept that RL-based strategies outperform handcrafted strategies which are manually tuned for a wide spectrum of application scenarios. After theoretically motivating our approach, we turn to the practical problem of how to learn optimal strategies for new application domains where no prior system or in-domain data are available. We propose to learn dialogue strategies by simulation-based RL, where the simulated environment is learned from small amounts of WOZ data. We therefore introduce a 5-step procedure:

1. Collect data in a WOZ experiment.
2. Use this data to construct a simulated learning environment using data-driven methods only.
3. Train a RL-based dialogue policy by interacting with the simulated environment. We compare this policy against a supervised baseline. This comparison allows us to measure the relative improvements over the WOZ strategies contained in the training data.
4. Evaluate the learned policy with real users.
5. Show that "bootstrapping" from WOZ data is a valid estimate of real HCI by comparing different aspects of the 3 corpora gathered so far: the WOZ study, the dialogues generated in simulation, and the final user tests.

It should be noted that these steps are not unique to the method introduced in this book, but most of the defined steps are required for any simulation-driven approach to strategy learning (though the last step is specific to our method). A main contribution of this book is that all these steps are now performed starting with a limited WOZ data set, and specific methods are introduced to build and validate the obtained simulations.

We apply this framework to optimise multimodal[1] Dialogue Management strategies and Natural Language Generation. In the first case we consider Dialogue Management and content selection as two closely interrelated problems for information seeking dialogues: the decision of *when* to present information depends on *how many* pieces of information to present and the available options for *how* to present them, and vice versa. We therefore formulate the problem as a hierarchy of joint learning decisions which are optimised together.

The second study describes a new approach to generating Natural Language in interactive systems. Natural Language Generation (NLG) addresses the problem of "how to say" an utterance, once "what to say" has been determined by the Dialogue Manager. We treat NLG as planning under uncertainty for information-seeking dialogue systems, where the strategy for information presentation and its associated attributes are incrementally selected using hierarchical learning. This hierarchical approach to DM and NLG has recently been explored by other researchers (Dethlefs and Cuayahuitl, 2010; Dethlefs and Cuayáhuitl, 2011), and a utility-based approach to NLG is discussed by van Deemter (2009a).

---

[1] "Multimodal" dialogue systems are those which do not only use speech input and output, but which also use other information modalities, such as graphics (as in our case), or gesture, gaze, facial expressions, and so on.

Our results in both studies show that RL significantly outperforms supervised learning (SL) when interacting in simulation as well as for interactions with real users. For optimising multimodal Dialogue Management, the RL-based policy gains on average 50-times more reward than the SL policy when tested in simulation, and almost 18-times more reward when interacting with real users. Users also subjectively rate the RL-based policy on average 10% higher. For optimising Natural Language Generation, the trained information presentation strategies significantly improve dialogue task completion, with up to a 9.7% increase (30% relative) compared to the deployed dialogue system which uses conventional, hand-coded presentation prompts.

One focus of this book is to optimise dialogue strategies with respect to real user preferences. A major advantage of RL-based dialogue strategy development is that the dialogue strategy can be automatically trained and evaluated using the same objective function (Walker, 2005). Despite its central importance for RL, quality assurance for objective functions has received little attention so far. In fact, the reward function is one of the most hand-coded aspects of RL (Paek, 2006). Clearly, automatic optimisation and evaluation of dialogue policies, as well as quality control of the objective function, are closely inter-related problems: how can we make sure that we optimise a system according to real users' preferences? This book is the first to explore learning with data-driven, non-linear objective functions. We also propose a new method for meta-evaluation of the objective function.

Note that chapters 4 to 8 are significantly revised, updated, and extended versions of material from (Rieser, 2008).

1.3 Structure of the Book

Chapter 2 (Background)

This chapter provides the reader with relevant background knowledge for the research. After introducing some general information about Spoken Dialogue Systems, we contrast different methods applied in research and industry to develop dialogue strategies. We show how these two approaches fail to meet the current challenges for strategy design and argue for the use of statistical methods. In particular, we propose the use of Reinforcement Learning, as it uses trial-and-error exploration with delayed rewards which, we argue, is a natural model for human dialogue.

Chapter 3 (Reinforcement Learning)

This chapter provides technical background on RL for dialogue strategy development and discusses simulation-based learning in particular. We also introduce the application domain.

Chapter 4 (Proof-of-Concept: Information Seeking Strategies)

He we develop the theoretical proof-of-concept that RL-based strategies outperform hand-coded strategies, which are tuned to the same objective function. We show this for a wide range of application scenarios, e.g. for different user types and noise conditions. This chapter also demonstrates how to apply simulation-based RL to solve a complex and challenging problem for information-seeking dialogue systems: which questions to ask the user, how many database search results to present, and when to present them, given the competing trade-offs between the length of the results list, the length of the interaction, the type of database, and noise in the speech recognition environment. In this chapter the reader will receive a deeper understanding of the principles and advantages of Reinforcement Learning for dialogue strategy learning. We also explain why data-driven simulation approaches are preferred over manually constructed simulated environments (as done here for the theoretical proof-of-concept).

Chapter 5 (A Bootstrapping Approach to Develop Reinforcement Learning-based Strategies)

This chapter introduces a 5-step procedure model to bootstrap optimal RL-based strategies for WOZ data. The resulting strategies are tailored to the application environment, do not require a working prototype system, and are optimised with respect to real user preferences. We also explain how we meet the challenges when learning from WOZ data. In particular, we introduce the problem of how to construct a simulated environment from limited amounts of WOZ data. We also discuss the fact that a WOZ study itself is a simulation of real HCI.

Chapter 6 (Data Collection in a Wizard-of-Oz Experiment)

Here we describe the experimental setup of the WOZ experiment. We explain which changes to the conventional WOZ method are necessary for strategy learning. In particular, we introduce an utterance distortion method in order to resemble noise conditions for real dialogue systems. In addition, we explore the "intuitive" strategies that were applied by our human wizards.

Chapter 7 (Building a Simulated Learning Environment from Wizard-of-Oz Data)

This chapter uses the WOZ data to construct a simulated learning environment. We therefore introduce methods suited to build and validate simulations from small amounts of data. In particular, we construct the action set and state space by exploring the wizards' actions. The user and noise simulations are obtained using frequency-based approaches. The objective function is a predictive model of user ratings obtained by a regression analysis, following the PARADISE framework of (Walker et al, 1997). We then train a RL-based dialogue policy by interacting with the simulated environment, and we compare this strategy against a baseline constructed by Supervised Learning. This comparison allows us to measure the relative improvements over the wizard strategies obtained from the training data.

Chapter 8 (Comparing Reinforcement and Supervised Learning of Dialogue Policies with Real Users)

In this chapter we evaluate the learned strategy with real users. We therefore develop a music-player dialogue system using a rapid development tool, where the learned strategy is implemented using a table look-up between states and learned actions. We report detailed results from the real user tests.

We also post-evaluate our overall "bootstrapping" approach by comparing different aspects of the 3 corpora gathered so far: the WOZ study, the dialogues generated in simulation, and the final user tests. We first evaluate whether strategies learned in simulation do transfer to tests with real users, and we also compare the experimental conditions of the different studies, where we discuss the noise model in particular. Furthermore, we explore whether the objective function used for learning is a realistic estimate of real user preferences.

Chapter 9 (Natural Language Generation)

This chapter further develops the methodology to encompass elements of policy learning for adaptive Natural Language Generation in spoken dialogue systems. This chapter shows that our method can quite easily be applied to new domains and tasks. We show how to develop a data-driven approach to content selection and structuring decisions in NLG, this time in the domain of a restaurant recommendation SDS. We also report results from evaluations with both simulated and real users. The real user evaluation shows that improved NLG can lead to significant improvements in overall dialogue system performance.

Chapter 10 (Conclusion)

Finally, we conclude by summarising the main contributions of this work. We also report on "lessons learned" to provide guidance for future researchers. For example, we discuss the data quantity and quality required for the proposed bootstrapping approach. In the final outlook of this chapter we discuss ongoing challenges for research in this field.

Part I
Fundamental Concepts

Chapter 2
Background

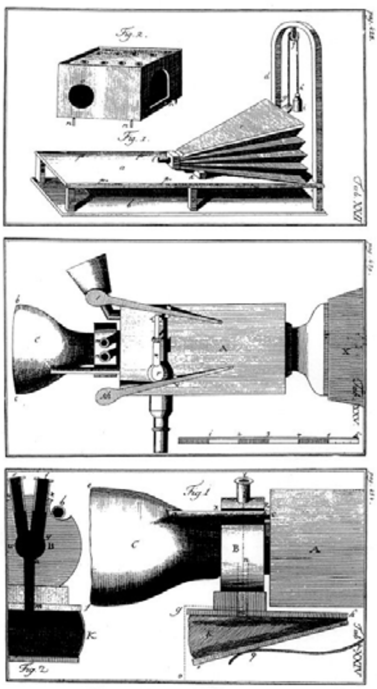

Fig. 2.1 Wolfgang von Kempelen's speaking machine, drawing from *Mechanism of Human Speech* (1791)

Mankind appears to be fascinated by the idea of talking with machines. The first attempts to produce human speech by machine were made in the second half of the 18th century. One of the best known examples is Wolfgang von Kempelen's speak-

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_2,

ing machine, as described in his book "Mechanism of Human Speech" [1] (1791), see Figure 2. Von Kempelen's machine was the first that produced not only some speech sounds, but also whole words and short sentences. Clearly, in order to converse with a machine in natural language more than just automatic sound production is necessary. Descartes even thought that it would never be possible to engage in dialogue with machines at all. In his book "Discourse on the Method of Rightly Conducting the Reason, and Searching for Truth in the Sciences" [2] (1637) he declares:

> "... but if there were machines bearing the image of our bodies, and capable of imitating our actions as far as it is morally possible, [...] they could never use words or other signs arranged in such a manner as is competent to us in order to declare our thoughts to others."

Today, however, it is arguable that such machines now exist, at least for limited application domains, due to major advances in the field of Human-Computer Interaction (HCI) and Spoken Dialogue Systems (SDS) . Still, human-machine dialogue is far from resembling the capabilities of human-human dialogue, as we discuss below.

2.1 Human-Computer Interaction

For computers, holding a conversation is difficult. Engaging in a conversation requires more than just technical language proficiency. When people engage in dialogue, they carry out a purposeful *activity* (Austin, 1962), a *joint action* (Clark, 1996), or a *language game* (Wittgenstein, 1953), which they know how to perform using their communicative skills. Dialogue behaviour is often formally described as a sequence of Speech Acts (SAs) (Searle, 1969). These SAs are organised into structural patterns in the field of Conversation Analysis, e.g. (Levinson, 1983; Sacks et al, 1974). Some of this behaviour follows standardised cultural conventions. For example, the fact that people greet each other is described as the standardised SA "adjacency pair" `greeting-greeting` (Levinson, 1983). Other behaviour is highly context-dependent. For example, in previous work we show that the way people ask for clarification is influenced by various contextual and environmental factors, such as dialogue type, modality, and channel quality (Rieser and Moore, 2005). In addition, people often engage in dialogue to solve a task together. Their behaviour is then driven by the goal of the task as well as by their "mental model" (Johnson-Laird, 1983) of the other dialogue participant.

Humans acquire these communicative skills over time, but for a dialogue system, they need to be developed by a dialogue designer. This usually is an expert who defines a *dialogue strategy* , which "tells" the system what to do in specific situations. The "dialogue strategy" is part of the Dialogue Manager (DM) which controls the

---

[1] Original title: *Mechanismus der menschlichen Sprache nebst Beschreibung einer sprechenden Maschine*

[2] Original title: *Discours de la méthode pour bien conduire sa raison, et chercher la verité dans les sciences*

behaviour of the system. Broadly speaking, a dialogue system has three modules, one each for input, output, and control, as shown in Figure 2.2. The input module commonly comprises Automatic Speech Recognition (ASR) and Spoken Language Understanding (SLU). The control module corresponds to the Dialogue Manager, which executes a dialogue strategy. The output module consists of a Natural Language Generation (NLG) system and a Text-To-Speech (TTS) engine. Usually, these modules are placed in a pipeline model (see Figure 2.2). The ASR converts the user's speech input (1) into text (2). SLU parses the text into a string of meaningful concepts, intentions, or Speech Acts (3). The Dialogue Manager maintains an internal state and decides what SA action to take next (4). This is what we call a dialogue strategy. For most applications the DM is also connected to a back-end database. In the output module, NLG renders the communicative acts (4) as text (5), and the TTS engine converts text to audio (6) for the user. Interested readers are referred to introductory texts such as (Bernsen et al, 1998; Huang et al, 2001; Jurafsky and Martin, 2000; McTear, 2004).

Human-Computer Interaction (HCI) is the study of interaction between people (users) and computers (such as dialogue systems). Human-machine dialogue differs from human-human dialogue in various ways. The most prominent features are the lack of deep language understanding and the lack of pragmatic competence (communicative skills) of the system. The lack of language understanding is due to errors introduced by less-than-perfect input processing (ASR and NLU), and the common use of shallow semantic representations. The lack of pragmatic competence is mainly due to the limited capabilities (often hand-coded heuristics) of the control module.

A substantial amount of recent work targets the problem of limited language understanding capabilities with so-called "error handling", e.g. (Bohus, 2007; Frampton, 2008; Skantze, 2007a), or "uncertainty handling" mechanisms, e.g. (Thomson and Young, 2010; Williams, 2006; Williams and Young, 2007a). This book addresses the problem of pragmatic competence: how to improve the communicative skills of a system by providing effective mechanisms to develop better dialogue strategies. In particular, this book explains how to automatically learn these skills from experience using simulated interactions, starting with a small experimental study of human behaviour. We now explain the conventional methods for strategy development.

2.2 Dialogue Strategy Development

There is a wide range of techniques to develop dialogue strategies, and techniques applied in industry are very different from the ones applied in research (Griol et al, 2010; Pieraccini and Huerta, 2005; Williams, 2008). These differences reflect the fact that academia and industry often pursue different objectives. Academic systems often aim to emulate human behaviour in order to generate 'natural' behaviour, whereas commercial systems are required to be robust interfaces in order to solve

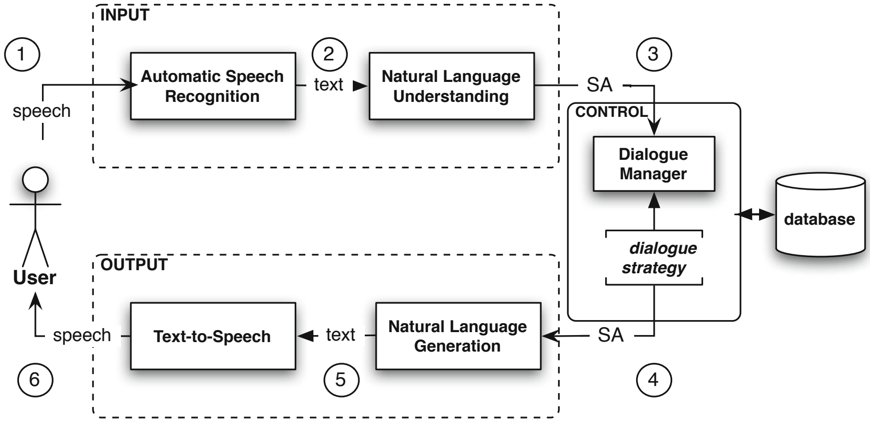

Fig. 2.2 Dialogue system architecture and processing pipeline

a specific task, see (Larsson, 2005; Pieraccini and Huerta, 2005) for further discussion.

In the following we first describe the general development cycle for dialogue strategies (which is commonly used in industry as well as in research). We then focus on two central aspects of this cycle, where techniques in research and industry differ widely: strategy evaluation/quality control and strategy implementation/formalisation. We later argue for a computational learning-based approach, where the standard development cycle is replaced by data-driven techniques.

2.2.1 Conventional Development Lifecycle

The development lifecycle for dialogue strategies is in many ways similar to the traditional software engineering approach. This type of dialogue strategy development is used in industry as well as in research projects (Bernsen et al, 1998; McTear, 2004). The lifecycle model includes a number of sequential stages (see Figure 2.3): requirements analysis and functional specification, design, implementation, testing and evaluation. In practise, the concrete realisation of the individual stages varies from system to system. Here we briefly summarise the major aspects for each stage.

In the initial planning stage, a so-called "requirements analysis" is performed: the system designer examines the use case of the system (e.g. the role and function of a system, user profiles, usage patterns), and the language requirements (e.g. vocabulary, grammars, interaction patterns). Sometimes an initial user survey, e.g. (Rieser, 2003; Wang et al, 2005), or an initial Wizard-of-Oz experiment, e.g. (Kruijff-Korbayová et al, 2005a), helps to specify further requirements. For commercial systems, the client also provides a set of functional specifications.

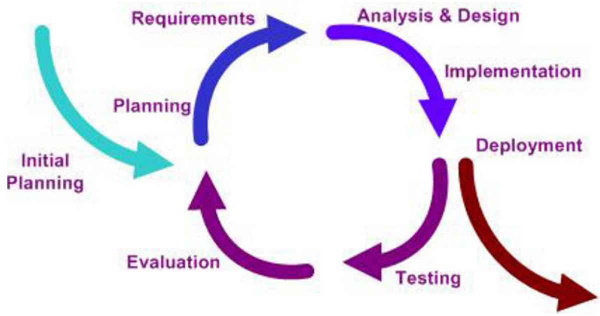

Fig. 2.3 Iterative development cycle used to develop dialogue systems

In the second stage an initial strategy is designed. The dialogue is commonly visualised using flow charts, also known as Task Hierarchical Diagrams (McTear, 2004), which describe all the possible choice points for dialogue tasks and sub-tasks as a finite state automaton.

After the dialogue strategy is defined on paper, it is implemented into a working dialogue system, for example using VoiceXML [3] (or other alternatives as discussed in Subsection 2.2.3), translating the design decisions into code.

In the final stage, the strategy is tested and evaluated. So-called "black-box" testing is frequently used to evaluate the system as a whole with reference to its functional specification, its performance, and its user acceptance. Further details are discussed in Subsection 2.2.2. The evaluation results and an error analysis are used to inform strategy re-design, and the cycle starts again.

In the next two Sections we further investigate the evaluation and implementation phase in more detail, where we compare methods used in industry with the ones used in research.

2.2.2 Evaluation and Strategy Quality Control

2.2.2.1 Quality Control in Industry

In industry the initial design is commonly motivated by guidelines and 'best practises' which should help to assure the system's usability (Paek, 2007). Various sources exist which define standards, guidelines and recommendations on what a "good" dialogue should be, e.g. (Balentine and Morgan, 2001; Balogh et al, 2004; Dix et al, 1998; Dybkjaer and Bernsen, 2000; Frostad, 2003; Lamel et al, 2000; Larson, 2003; Nass and Barve, 2005; Shneiderman, 1997; Weinschenk and Barker, 2000). However, the quality of the given advice is questionable (Paek, 2007). For

---
[3] `http://www.voicexml.org/`

example, the listed "do's and don'ts" are not substantiated by rigorous experimental design, and it is also not clear how well they generalise. Furthermore the given advice is sometimes very vague (e.g. "Reduce short term memory" (Shneiderman, 1997)), and may contradict each other (e.g. "Be brief" vs. "Avoid ambiguity"). It is up to the designer to translate these statements into a dialogue strategy.

The final system is released into deployment if it is "good enough" as evaluated by some metrics. For industrial applications this criterion is defined by Return-On-Investment (ROI), which is the ratio of money gained or lost on an investment relative to the amount of money invested (Paek, 2007; Pieraccini and Huerta, 2005). In general, the development costs for dialogue strategies are high. In particular, user testing is time, labour, and cost intensive. Thus, extensive evaluation is often set aside in practise. In software engineering this dilemma of insufficient testing was first described by Sneed and Merey (1985) as the "vicious square". According to this theory, the overall productivity of a software project is defined by quantity, costs, duration, and targeted quality. If costs, duration, and quantity are limited by ROI, software quality and software testing will need to be reduced.

2.2.2.2 Evaluation Practises in Academia

Dialogue strategies developed in academia are usually extensively tested against some baseline in order to make scientific claims, e.g. by showing some significant differences in system behaviour. Research has exerted considerable effort and attention in devising evaluation metrics that allow for comparison of disparate systems with varying tasks and domains: see (Paek, 2007) for an extensive survey. In the rest of this section we discuss the PARADISE framework in more detail. While there are many other evaluation metrics, e.g. (Hartikainen et al, 2004; Paek, 2001) etc., PARADISE has emerged as a "de-facto standard" (Möller et al, 2007), along with the SASSI framework by (Hone and Graham, 2000). These two evaluation frameworks serve different purposes. While SASSI (Subjective Assessment of Speech System Interfaces) is a framework for designing user questionnaires, the main purpose of PARADISE (PARAdigm for DIalogue System Evaluation) is to construct a data-driven model for automatic dialogue evaluation.

Furthermore, the questionnaire used in the original PARADISE studies (Walker et al, 1997, 2000) is also widely used by other subsequent dialogue studies, e.g. (Frampton and Lemon, 2006; Hajdinjak and Mihelic, 2006; Hof et al, 2006; Lemon et al, 2006a; Quarteroni and Manandhar, 2008). The questions used in PARADISE target different dimensions, as listed in Table 2.1. These dimensions have been criticised for being "arbitrary and based on intuition" (Paek, 2007). The SASSI framework (Hone and Graham, 2000), in contrast, allows a more principled approach to design questionnaires. In SASSI six main factors are identified which determine the user's perception of the dialogue. We argue that in fact the questions in PARADISE and SASSI measure similar underlying factors/ dimensions. For a direct comparison see Table 2.1.

In the next Subsection we describe the PARADISE framework in more detail.

SASSI	PARADISE
System Response Accuracy: User's perceptions of the system as accurate and doing what they expect.	**Expected Behaviour**: Did the system work the way you expected it to in this conversation?; **ASR performance** In this conversation, did the system understand what you said?
Likeability: User's rating of the system as useful, pleasant friendly.	**Comparable Interface**: In this conversation, how did the system's voice interface compare [to other systems]?; **Future Use**: From your current experience with using the system [...], do you think you'd use the system regularly [...]
Cognitive Demand: The perceived amount of effort needed to interact with the system and feelings arising from this effort.	**Task Ease**: In this conversation, was it easy to find the information you wanted?; **TTS performance**: Was the system easy to understand in this conversation?
Annoyance: User's rating of the system as repetitive, boring, irritating and frustrating.	**System Response**: How often was the system sluggish and slow to reply to you in this conversation?
Habitability: The extent to which users knew what to do and what the system was doing.	**User Expertise**: In this conversation, did you know what you could say at each point of the dialogue?
Speed: How quickly the system responded to user inputs.	**Interaction Pace**: Was the pace of interaction appropriate in this conversation?

Table 2.1 Comparison of dimensions for user satisfaction according to the SASSI and PARADISE questionnaires

2.2.2.3 The PARADISE Evaluation Framework

PARADISE is a widely used framework for automatic dialogue evaluation introduced by (Walker et al, 1997, 1998b, 2000). The main idea behind PARADISE is to estimate subjective user ratings (obtained from questionnaires) from objective dialogue performance measures (such as dialogue length) which are available at system runtime. (Walker et al, 1997) propose to model "User Satisfaction" (US) using multiple linear regression (see Equation 2.1). User Satisfaction is calculated as the arithmetic mean of nine user judgements related to different quality aspects (see Table 2.1), which are rated on Likert scales . A likert scale is a discrete rating scale where the subject indicates his/her level of agreement with a statement (e.g. from "strongly agree" to "strongly disagree").

The input to the regression analysis consists of two main constituents: A parameter related to task success (either the coefficient κ calculated from an external annotation of correctly identified concepts, or a direct user judgment on perceived task success), and of additional interaction parameters measuring dialogue efficiency and quality C_i. These parameters can include runtime features such as the dialogue duration, the number of system and user turns, a mean recognition score, as well as the number of time-out prompts, barge-ins, recogniser rejections, help requests and cancel attempts by the user, and so on (see (Möller, 2005b) for a discussion of other possible runtime features). The multivariate linear regression analysis is carried out with the input parameters κ and C_i as the independent (predictive) variables, and the mean user judgment US as the dependent (target) variable. The regression determines the weighting coefficients α and w_i, of the linear prediction function as in Equation 2.1, where $N(\cdot)$ is a normalisation function.

$$\underbrace{US}_{subjective} = \underbrace{\alpha \times N(\kappa) - \sum_{i=1}^{n} w_i \times N(C_i)}_{objective} \tag{2.1}$$

The major strength of this framework is that the resulting model can be used to predict subjective user ratings automatically from objective runtime features. Once the PARADISE regression model is determined from data by a user study, the expected User Satisfaction for all future dialogues generated by the system can be calculated automatically.

2.2.2.4 Strategy Re-Implementation

After testing and evaluation, an error analysis is performed and the results are then used to re-design the strategy. However, there is no framework which describes how evaluation results are best transferred into code. For example, the three features that consistently appeared among the top predictive factors for the PARADISE model are mean recognition score, task completion, and the percentage of recognition rejections, e.g. (Möller et al, 2007; Skantze, 2005; Walker et al, 2000; Williams and

Young, 2004a). Unfortunately, this is not the kind of insight which leads to improved strategies, as most system designers probably already know that improving speech recognition (either in absolute terms or by user perception) improves user satisfaction. The question therefore remains how strategy evaluation can directly lead to improved strategies.

2.2.3 Strategy Implementation

There are many different ways in which a dialogue strategy can be implemented. The approaches listed in Table 2.2 do not represent an exhaustive list of the techniques available.[4] It does however, give some idea of the wide range of the possible frameworks available. The interested reader is referred to the references in Table 2.2.

2.2.3.1 Implementation Practises in Industry

Most commercial systems rely on Finite State Automata (FSA) controlled by menus, forms, or frames.[5] The most common applications are form filling dialogues, information retrieval, transactions and services (Pieraccini and Huerta, 2005). Using the finite-state model, dialogue strategies can be rapidly prototyped, tested and debugged. Furthermore, FSAs are useful for small, well-defined tasks, and in situations where speech recognition performance may be relatively low (e.g. over a telephone line) as they allow small sub-grammars to be defined which facilitate robust recognition (e.g. digit recognition).

However, this development methodology is limited by the fact that every change in the conversation must be explicitly represented by a transition between two nodes in the network. Dialogue strategies designed as FSA are based on hand-crafted rules which usually lack context-sensitive behaviour, are not very flexible, cannot handle unseen situations, and are not reusable from task to task. Furthermore, FSA easily become intractable for more complex tasks and cannot model complex reasoning.

---

[4] Note that there is no agreed typology for classifying dialogue strategy implementations. The one used in this book follows Lemon (2006); McTear (2004).

[5] Note that this is not an absolute classification. There are also research systems which use FSA, e.g. (Alexandersson and Reithinger, 1995; Okamoto et al, 2001; Rieser, 2003).

Domains	Architectures and Tools	Dialogue control
Form filling e.g. train information system (Aust et al, 1995), flight booking COMMUNICATOR systems (Bennett and Rudnicky, 2002)	*Finite State Automaton* e.g. VoiceXML, CSLU toolkit	*menus, forms, frames* e.g. (Wang et al, 2005)
Information seeking e.g. music search (Varges et al, 2006)	*Information State Update* e.g. TRINDIKIT (Larsson and Traum, 2000), DIPPER (Bos et al, 2002), DUDE (Lemon and Liu, 2006)	*tree sub-structures* e.g. task agenda (Xu and Rudnicky, 2000), activity trees (Lemon et al, 2001)
Problem-solving e.g. TRIPS/TRAINS (Allen and Ferguson, 2002)	*BDI agents* e.g. COLLAGEN (Rich and Sidner, 1998)	*logical inference* e.g. (Grosz and Sidner, 1986), (Allen and Litman, 1987), (Sadek et al, 1996), (Blaylock and Allen, 2005), (Steedman and Petrick, 2007)
Tutorial and educational dialogue e.g. (Litman, 2002; Pon-Barry et al, 2004)		
Intelligent assistants e.g. COMPANIONS project (Benyon and Mival, 2007)		

Table 2.2 Frameworks conventionally used for dialogue strategy development

2.2.3.2 Implementation Practises in Academia

Most research systems to date have been based either on planning with logical inference, e.g. (Blaylock and Allen, 2005; Steedman and Petrick, 2007), or they are implemented in the "Information State Update" (ISU) approach using frames or tree sub-structures as control mechanism, e.g. (Larsson and Traum, 2000; Lemon et al, 2001). More recently, statistical systems using machine learning approaches have become more prevalent, for example (Griol et al, 2008; Henderson et al, 2008; Thomson and Young, 2010; Young et al, 2007, 2009), and see (Frampton and Lemon, 2009) for a survey.

Planning approaches are mostly used for complex tasks, like collaborative problem solving, intelligent assistants, and tutorial dialogues. ISU-based systems are used for a variety of applications with different complexity (see Table 2.2 for references). Both approaches have an higher expressive power than simple FSA, and can lead to more sophisticated (e.g. context-dependent) strategies. On the other hand, these systems are harder to maintain and debug. Building these types of systems requires (linguistic) expert knowledge, and the encoded strategy has to be manually tailored to a specific application and is not reusable. Furthermore, hand-tuning these complex strategies requires specialised knowledge of linguistic representation. Hand-tuning a simple FSA (as used in industry) only requires software engineering skills. Hence, the dialogue frameworks developed in research are often too costly to be applied commercially, though see (Griol et al, 2010; Pieraccini et al, 2009; Williams, 2008) for recent discussions of the use of statistical approaches in industry.

2.2.4 Challenges for Strategy Development

How can this chasm be bridged? Is there a third option which can meet the challenges for both cost-effective industrial speech interfaces and the advanced dialogue agents of academic research? What requirements does it have to meet?

In the following we discuss why standard techniques are not suited to meet the future challenges described in Zue (2007). Zue calls the dialogue system of the future an "*organic interface*", that can learn, grow, re-congure, and repair itself.

First, good strategies have to be more *robust* towards unseen events (Zue, 2007). The techniques outlined above require the manual specification of rules which define an action for all possible dialogue situations. Exhaustive enumeration, encoding, and maintenance of strategies becomes increasingly complex with growing complexity of the application. It is not practically possible for the designer to anticipate all the possible situations of a dynamic environment. Thus, "organic" interfaces will need a strategy which is able to *generalise to unseen events*.

Second, good strategies should be *context sensitive* (Zue, 2007). Most current strategies make use of hand-coded thresholds in order to react to changes in the context. For example, thresholds are used to make strategies sensitive to the number of retrieved database items, e.g. (Varges et al, 2006), or to the confidence scores returned from ASR, e.g. (Bohus and Rudnicky, 2005b). Most of the time, these thresholds stay fixed the whole dialogue. For strategies to be truly context-sensitive the thresholds would need to be re-defined in each context – a task which becomes increasingly complex, especially if multiple thresholds are applied. In addition, these thresholds have to be redefined in the same costly manner when the system is transferred to another application environment. Thus, while current threshold-based strategies are usually only crudely adapted to an overall state, "organic interfaces" should *dynamically adapt* to every possible system context.

Third, good strategies need to more *adaptive* to the application environment (Zue, 2007). For example, the dialogue should be adaptive to channel noise, different user behaviour, user preferences and user types. In order to design adaptive strategies the system developer has to foresee the requirements of the respective conditions and manually specify according strategies. Ideally, "organic interfaces" would be able to *automatically adapt* to different situations.

Therefore, Zue (2007) concludes that "organic interfaces" will be new interfaces which implement strategies which automatically *learn* adaptive behaviour, in contrast to the static standard techniques described in Section 2.2.3 above.

Current research has turned to automated dialogue strategy learning using statistical machine learning techniques, e.g. (Henderson et al, 2008; Levin and Pieraccini, 1997; Rieser and Lemon, 2011; Thomson and Young, 2010; Williams and Young, 2007a; Young, 2000) These techniques overcome many of the deficits of the conventional methods, such as a data-driven development cycle, a precise mathematical model for optimisation, possibilities for generalisation to unseen states, and reduced development and deployment costs for industry (Lemon and Pietquin, 2007).[6] However, statistical learning techniques for dialogue strategies, especially Reinforcement Learning, are also criticised for not being suitable for commercial development (Paek, 2006). In particular, (Paek, 2006) criticises the large amounts of data that are needed, and complains that the learned policy is a "black box" which cannot be controlled by the system designer. In the course of this book we will discuss this (and other) criticisms. In the final Chapter (section 10.2.2) we will summarise the arguments. A major advantage of this new statistical approach is that it introduces a principled scientific method for improving dialogue strategy design, whereas the previous hand-coded approaches were mainly based on the designer's intuition.

We now present different Machine Learning paradigms and discuss which are best suited for dialogue strategy development.

[6] Note that this research combines efforts from academia as well as from industry. The first statistical dialogue systems were actually developed in the context of industrial research (Levin and Pieraccini, 1997; Walker et al, 1998a).

2.3 Literature review: Learning Dialogue Strategies

2.3.1 Machine Learning Paradigms

Machine Learning (ML) can be defined as follows. Given a specific task to solve, and a class of functions F, learning means using a set of observations, in order to find $f^* \in F$ which solves the task in an optimal sense. This entails defining a cost function $C : F \rightarrow \Re$ such that, for the optimal solution $\forall f \in F, f^*, C(f^*) \leq C(f)$ (no solution has a cost less than the cost of the optimal solution). The cost function C is an important concept in learning, as it is a measure of how far away we are from an optimal solution to the problem that we want to solve. Learning algorithms search through the solution space in order to find a function that has the smallest possible cost (or alternatively the maximal utility). Several textbooks offer a comprehensive introduction to Machine Learning, e.g. (Bishop, 2006; Ghahramani, 2004; MacKay, 2003; Witten and Frank, 2005)

In general, there are three major learning paradigms, each corresponding to a particular abstract learning task: Supervised Learning, Unsupervised Learning and Reinforcement Learning. We now briefly define each paradigm, following Ghahramani (2004).

In *Supervised Learning* (SL), we are given a set of example pairs/labelled data points $(x,y), x \in X, y \in Y$ and the aim is to find a function f in the allowed class of functions that matches the examples. In other words, we wish to *infer* the mapping implied by the data; the cost function is to reduce the mismatch between our mapping and the data. The goal is to find a model which mimics the data as close as possible, while still being general enough to classify/predict unseen events well.

In *Unsupervised Learning* (US) we are given some data x, and the cost function to be minimised can be any function of the data x and the (unknown) target output. In contrast to SL, no target outputs/input labels are given. In a sense, US can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise. Tasks that fall within the paradigm of unsupervised learning are in general *estimation* problems; the applications include clustering, or the estimation of statistical distributions.

Reinforcement Learning (RL) is *sequential* decision making, where the RL agent interacts with its environment (Sutton and Barto, 1998). The environment is defined as:

> "anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment" (Sutton and Barto, 1998, p.53)

For dialogue strategy learning the simulated environment can include the (simulated) user, channel noise, the back-end database and other components of the dialogue system, such as ASR, NLU, and TTS. At each point in time t, the agent performs an action a_t and the environment generates an observation o_t and an instantaneous cost c_t (here also called "rewards"), according to some (usually unknown) dynamics. The goal is then to discover a policy for selecting actions that minimises some measure of a long-term cost and maximises the expected cumulative utility (also known as 'final reward'). This is usually done by trial-and-error search methods. A detailed introduction to RL is given in Section 3.2.

To date, different Machine Learning approaches have been applied to automatic dialogue management:

- Supervised approaches, which learn a strategy which mimic a given data set;
- Approaches based on decision theory, which are supervised approaches in the sense that they optimise action choice with respect to some *local* costs as observed in the data. In contrast to SL they explicitly model uncertainty in the observation;
- Reinforcement Learning-based approaches, which are related to decision theoretic approaches, but optimise action choice *globally* as a sequence of decisions.

We now explain these techniques in more detail and review recent applications for each ML paradigm.

2.3.2 Supervised Learning for Dialogue Strategies

Supervised learning follows a "learn-by-example" approach which learns a mapping between given inputs and outputs from a fixed data set. In the following we review three examples of supervised dialogue strategy learning.

The research of (Lane et al, 2004; Ueno et al, 2004), for example, adapts dialogue strategies to various user and situation models via example-based learning. The training corpus is gathered using the following setup: a set of possible system responses is displayed on a screen while the user interacts with the system. For each system turn, the user selects the response that they think is most suitable in the current situation.The learned strategy chooses the action which is selected most often by the users. Note that this method is similar to Active Learning, where the human annotator selects labels for the most informative cases (Cohn et al, 1994; Seung et al, 1992). However, one has to assume that users are not experts in dialogue strategy design. As a consequence, the annotations can be considered less than optimal. In addition, this type of user assisted design has high costs, compared to strategy design by an expert.

A different approach to "human assisted design" is introduced by Okamoto et al (2001). They use a Wizard-of-Oz (WOZ) study (see Section 3.3.1) for data collection, where the human wizard is free to choose from a predefined set of utterances. They assume that learning an "average" strategy of decisions taken by the human wizard will result in a good dialogue strategy. Again, the wizard is not an expert in how a machine should ideally react in a certain situation. In general, human decisions often serve as "gold standard" for other tasks such as automatic summarisation (Teufel and van Halteren, 2004), or automatic annotation. However, human behaviour cannot be viewed as gold standard in the context of dialogue strategy design, as Human-Machine Interaction is fundamentally different to Human-Human Interaction (as argued in Section 2.1). We will provide some examples of sub-optimal wizard behaviour in Chapter 6.4. In addition, (Okamoto et al, 2001) report that SL methods applied to WOZ data suffer from data-sparsity. (Note that WOZ studies are costly and therefore usually result in limited amounts of data.)

In order to circumvent the data problem, (Filisko and Seneff, 2005, 2006) generate training data by interaction with simulated users, based on work by Chung (2004a). A simulated user generates text-based utterances, which are the converted to sound. An ASR system is used to recognise what was said, while the dialogue manager explores different error handling strategies. They use SL to select the strategy which is most likely to resolve the error in the next turn, as labelled in the data.

In sum, SL approaches optimise some point-based decision, i.e. they learn to estimate the most successful action in a certain context/dialogue state as observed in a fixed data set, where "success" can be label indicating whether the error was resolved in the next turn (Filisko and Seneff, 2005, 2006), a rating assigned by human judges (Lane et al, 2004; Ueno et al, 2004), or the action taken most frequently by the human wizard (Okamoto et al, 2001).

However, SL approaches have two short-comings with respect to dialogue strategy learning: First, they do not model uncertainty in what was recognised, which is one of the major characteristics of human-machine dialogue (see Section 2.1). Second, they do not model dialogue as a sequence of actions, but are only based on local point-wise estimates. Third, they only mimic behaviour observed in a fixed corpus and no new strategies can be explored. The first short-coming is corrected by approaches using Decision Theory as discussed in the next Section. The other two are corrected by Reinforcement Learning, as discussed in Section 2.3.4.

2.3.3 Dialogue as Decision Making under Uncertainty

Dialogue was first described as *decision making under uncertainty* by Paek and Horvitz (Paek and Horvitz, 1999, 2000, 2003). Similar to SL, this approach is based on some local cost function, defining a mapping states and actions, which is here called *utility*. In addition, this approach also explicitly models the uncertainty in the observed state. In this framework the agent selects the action $A = a$ that maximizes expected utility, $EU(a|o)$, where o are observed events. Action selection is guided by the following optimisation:

$$A = \operatorname{argmax} EU(a|o) = \operatorname*{argmax}_{a} \sum_{s} P(S = s|o) \times utility(a,s); \qquad (2.2)$$

where $utility(a,s)$ expresses the utility of taking action a when the state of the world is s. The utility function is trained via "local" user ratings. Users rate the appropriateness of an action in a certain state via a GUI while they are interacting with the system (similar to (Lane et al, 2004; Ueno et al, 2004) for SL). Paek and Horvitz apply this framework to error-handling sub-strategies.

Bohus (2007) and Skantze (2007a) follow a similar utility-based approach, also applying it to error handling. The work of Bohus et al (Bohus, 2007; Bohus and Rudnicky, 2005b; Bohus et al, 2006) follows a similar approach, but derives the utility function from (post-annotated) dialogue data. In this work, binary logistic regression is used to determine the costs between task success and various types of understanding errors. Different regressions may then be calculated in different dialogue states, resulting in more dynamic behaviour than simple threshold setting (cf. discussion in Section 2.2.4). In addition, Bohus applies this framework to learn whether to *reject* or *accept* a user input.

Recent work by Skantze Skantze (2007a,b) extends this approach and learns whether to *reject*, *accept*, *display* understanding, or *clarify* a user input. Skantze learns the cost for these actions from data gathered with the HIGGINS dialogue system (Edlund et al, 2004).

In sum, this approach to dialogue as *decision making under uncertainty* has been applied to optimise local error handling strategies i.e. how to detect and recover from an ASR error within in a certain local context, given some uncertainty. However, this framework doesn't consider what action is best *in the long run*, i.e. how this local decision will contribute to a successful dialogue in the light of how the dialogue proceeds after taking this action. For real-life problems, multivariate functions commonly have many local minima and maxima. What is identified to be best in the current situation, may not be optimal for the overall goal of the dialogue (e.g. to efficiently solve a task). For example, results from a corpus study by Bohus and Rudnicky (2005a) show that there is a discrepancy of up to 51.5% between what is mis-recognised by the system and what is corrected by the users. Similar observations are also reported by (Acomb et al, 2007), where 13.2% of the confirmed utterances are mis-recognised. In other words, users may leave errors *locally* uncorrected, which then *globally* leads to task failure.

2.3.4 Reinforcement Learning for Dialogue Strategies

In contrast to the above approaches, Reinforcement Learning treats dialogue strategy learning as a *sequential* optimisation problem, leading to strategies which are *globally* optimal (Sutton and Barto, 1998).[7]

Similar to Decision Theory, uncertainty can be explicitly represented in RL. Stochastic variation in the user response is represented as transition probabilities between states and actions using Markov Decision Processes (MDPs), which we introduce in Section 3.2.1. Furthermore, dialogue strategies need to be robust to "noisy" observations, for example speech recognition errors introduced by ASR. Therefore, the problem is represented as Partially Observable Markov Decision Process (POMDP), which we introduce in Section 3.2.1.2. While being conceptually

[7] Note that RL is the global optimisation technique which is most frequently applied to dialogue strategy learning. However, others do exist. For example, work by Toney (2007); Toney et al (2006), for example, explores the use of Genetic Algorithms for dialogue strategy learning.

more attractive that MDPs, POMDP-based SDS still suffer from high computational power, memory, and storage requirements of the hardware used, thus limiting the complexity of the systems currently being deployed.

We now review three RL-based systems which address one of the most urgent problems for RL-based strategy development: while being theoretically most attractive for dialogue strategy learning, RL needs substantial amounts of data to learn reliable strategies (also see discussion in Section 3.2.2.3). This is due to the following facts: If a situation was not seen in the training data (i.e. is unseen) the agent has to "guess" what to do, leading to less robust strategies. Furthermore, it leads to unreliable estimates of the expected utility (reward) of an action if state-action pairs are visited very infrequently. This is due to the fact that the dialogue learner interacts with a stochastic environment: every time the agent visits a state and executes an action, the environment (i.e. the user, ASR, etc.) might react differently. Thus a state action-pair needs to be visited several times in order to obtain reliable estimates, which is less likely when learning from limited data. Several approaches have been suggested to overcome the data sparsity problem for RL.

One technique addressing the data sparsity problem, is to limit the number of possible system states to the size of the available data set. For example Litman et al (2000); Singh et al (2002) use RL to optimise initiative and confirmation (sub-) strategies for the NJFun system. They learn these strategies from a fixed data set comprising 311 dialogues from a user study with 54 subjects. In order to learn reliable strategies from this data, Singh et al keep the policy space as small as possible. The strategy is limited to 42 possible states, where only two actions are made available in each state. They report that only 8 (19%) states are visited less than 10 times in the training data. The evaluation results of Litman et al (2000); Singh et al (2002) are encouraging: results with real users show significant increase in task completion rate from 52% to 64%. However, learning with limited state spaces still requires "a fair amount" data. Note that, Tetreault et al (2007) introduce a measure based on strategy convergence which helps to determine how much data is needed relative to the size of state-action set. However, learning with limited state spaces is in general not desirable, as only very simple strategies can be learned (and for very simple strategies hand-coded strategies might perform equally well). Another approach to handle the data problem is to summarise the state space in order to generalise to unseen situations as described below (also see Section 3.2.2.3).

The work of Henderson et al (2005, 2008) addressed a much more complex problem using RL, which is represented by 10^{386} states and 74 actions (resulting in possible $74^{10^{386}}$ policies). They therefore learn from a much larger data set of 2331 dialogues and over 103000 turns, taken from the COMMUNICATOR corpus (Walker et al, 2002b). However, they find that state frequencies in the data set follow a Zipfian (i.e., long-tailed) distribution, with 61% of the system turns having states that only occurred once in the data. Henderson et al (2008) address the data sparsity problem using two different techniques. First, linear function approximation is applied. Linear function approximation learns linear estimates between state-action pairs and expected reward values (Sutton and Barto, 1998), and thus generalises to states which are not in the training data (also see Section 3.2.2.3). In addition, Henderson et al (2008) apply a novel hybrid approach combining Supervised and Reinforcement Learning. SL is used to model which action to take in portions of the state space where data is not sufficient. RL is used to choose between the remaining states. The results show that the hybrid RL/SL policy outperforms the strategies which are based on RL or SL only. The SL policy is a very challenging baseline as it mimics a 'multi-version' system of 8 well-engineered COMMUNICATOR systems (Walker et al, 2002b). Multi-version systems are known to remove errors made by any one system that are not shared by most of the other systems.[8] Still, RL is able to improve upon SL by *exploring* different actions and evaluating how good they are in the long run (i.e. with respect to all subsequent actions which are likely to follow), rather than just choosing the locally best action according to the 8 systems. This result also illustrates that hand-coded strategies are very unlikely to be globally optimal: even the very "essence" of 8 expert systems manually designed by a human developer, is still sub-optimal. RL-based strategies, in contrast, are provably optimal. However, the exploration of new strategies as done by (Henderson et al, 2005, 2008) is very limited as they learn from a fixed data set. In addition, this approach relies on a substantial amount of initial training data, where annotated dialogue data is still scarce (Lemon and Pietquin, 2007).

A radically different approach to solve the data sparsity problem was introduced by Eckert et al (1997, 1998); Levin et al (2000). In this approach, the RL strategy is trained while interacting with a simulated dialogue environment, including a simulated user (cf. (Filisko and Seneff, 2005, 2006) for SL). This approach allows artificial expansion of a small amount of initial training data in a two-stage approach: a statistical learning environment is first trained on a limited amount of dialogue data and the simulated environment is then used to generate additional dialogues by interacting with the dialogue policy. However, the simulation-based approach assumes the presence of a small corpus of annotated in-domain dialogues. In addition, the quality of the learned strategy depends on the quality of the simulated learning environment. We address these problems in the course of the book .

Finally, new techniques have been explored very recently for rapid learning from small amounts of data (Gasic and Young, 2011; Pietquin et al, 2011a,b), without the use of user simulations. These methods may ultimately allow online learning of dialogue strategies during interaction. For example, (Pietquin et al, 2011a) learn sparse representations of the value function by sample efficient off-policy learning using Kalman Temporal Differences, also see Section 3.2.2.2.

2.4 Summary

This Chapter argues for the use of Reinforcement Learning (RL) to develop and optimise dialogue strategies. We first introduced a general discussion of Human-Human and Human-Computer Interaction (HCI). Dialogue systems for HCI require

[8] For example, multi-expert/ensemble-based learning has also shown to lead to improved results for parse selection (Osborne and Baldridge, 2004)

a dialogue strategy which “tells” the system what to do. We then presented the conventional lifecycle model for dialogue strategy development and discuss general differences between practises applied in industry and research. We concluded that the current challenge for dialogue development are strategies which are robust, adaptive, context-sensitive and cheap to develop. For this reason, statistical machine learning approaches have become attractive. We compare three different learning paradigms which are currently applied to strategy development: Supervised Learning, Decision Theoretic approaches, and Reinforcement Learning.

Having summarised the general background to our work, the next chapter deals with the specifics of Reinforcement Learning approaches, Dialogue simulation methods, and the application domain of our case-study.

Chapter 3
Reinforcement Learning

Fig. 3.1 Peter Rabbit (Potter, 1902): An agent receiving rewards

In this Chapter we summarise and discuss the main arguments for the use of RL in dialogue development, and introduce the reader to the technical details of this approach. We provide the reader with more background knowledge about the basic principles and ideas behind RL, and we relate these principles to basic characteristics of dialogue interaction. In particular, we argue that dialogue is *temporal* and *dynamic* in nature (Levin et al, 2000; Walker, 1993), which makes learning with *delayed rewards*, *instructive feedback*, and *exploration* particularly attractive. For

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_3,

the purpose of illustration we take the metaphor of task-oriented dialogue as a game of chess, since playing board games has similar temporal and environmental characteristics to dialogue (cf. Wittgenstein (1953)'s language games) which requires strategic planning under uncertainty.[1]

3.1 The Nature of Dialogue Interaction

In the following we argue that dialogue is *temporal* and *dynamic* in nature (Levin et al, 2000; Walker, 1993), and discuss properties of the previously mentioned statistical machine learning approaches (Supervised Learning, Decision Theory and Reinforcement Learning), see Section 2.3.

3.1.1 Dialogue is Temporal

Dialogue is *temporal* in the sense that how 'good' an action is depends on how the dialogue progresses further. Taking an action affects the state of the dialogue and thereby affects the options and opportunities available at later times. Thus, action choice requires foresight and long-term planning with respect to the delayed consequences of actions as specified by to the dynamics of the environment. As such, we cannot present correct input/output move pairs of ideal dialogue strategy behaviour. We often have data containing annotations of how good the overall performance of a specific dialogue was (e.g. task success, or user scores), but we don't have any indication about how good a single action was. In other words, it is hard to tell how things should have been exactly done in a specific situation, but we can tell whether the dialogue was successful/ satisfying overall.

Thus, Supervised Learning is not a suitable candidate for dialogue strategy optimisation (Levin et al, 2000). Nevertheless, SL has potential in the context of "multi-expert" learning, as shown by Henderson et al (2005, 2008) (see Section 2.3.4).

Local Decision Theory also does not account for the temporal nature of dialogue. DT produces dialogue strategies which are optimal in a certain context, but do not account for any long-term consequences. In the context of error handling (cf. (Bohus, 2007; Skantze, 2007a)) one might argue that to recover from an ASR error is a local problem. However, error recognition, i.e. the decision whether to engage in an error handling sub-dialogue or progress with the overall task, also affects long-term goals such as dialogue efficiency and task success. Even though Bohus (2007) and Skantze (2007a) specify their cost function with respect to a final outcome measure

[1] Note that chess is a competitive game, whereas dialogue is described as "joint action" (Clark, 1996) or "collaborative problem solving" (Blaylock and Allen, 2005). However, this difference will mainly affect the way actions are rewarded. It is still instructive to take chess as a metaphor to describe the general underlying principles for dialogue as a long-term planning process. Current research also explores non-cooperative dialogue (Plüss, 2010)

(task completion), they do not model the consequences of local decisions on further dialogue behaviour. Results by Frampton and Lemon (2008) show that good repair strategies are based on knowledge about the dialogue history, as well as the further course of the dialogue. In analogy with the chess-playing example, the Decision Theoretic approach corresponds to a strategy which only improves short-term tactics, but does not pay attention to how the move changes the overall long-term configuration on the chess board.

Reinforcement Learning, in contrast, models the problem as sequential decision process with long-term planning. RL is based on the principle of *delayed rewards*, i.e. the effects of an action may only become fully apparent at the end of the dialogue. Delayed rewards ensure that the value of an action in a given state is always defined with respect to the final outcome of a task, and thus helps to avoid local minima. For example, a good long-term tactic for a chess player might be to sacrifice a pawn in order to change the chess board constellation which then allows execution of a strategy which is promising in the long-run.

3.1.2 Dialogue is Dynamic

Dialogue being *dynamic* describes the fact that dialogue takes place in interaction with a stochastic environment, where conditions change frequently (e.g. the level of noise) or a dialogue partner reacts differently than predicted in a certain situation. This characteristic requires a strategy which is robust to unseen states.

In Supervised Learning the learner has to be explicitly 'told' what to do in a specific situation by presenting as many examples as possible. This is called learning via *instructive feedback*: telling the learner what is right or wrong in given situation. Once the system deviates from the dialogue in the corpus, there is no way of finding the correct supervision for the rest of the transaction.

In the chess-playing example, if somebody explains how to play chess, this person could never think of all possible board constellations and tell exactly what to do in each situation. Once the basic rules of the environment (the game) are known, an experienced player dynamically adapts his strategy to the current constellation, the progress of the game, the behaviour of the other player, and so on. Furthermore, a chess player will not consider the value of a single action choice, but will always plan his/her moves by "thinking ahead", i.e. as a sequence of action choices. Good players usually develop this skill by exploring different strategies over time (getting rewarded by winning and 'punished' by losing), and start to exploit this knowledge at a later point. Similarly, language learners improve their communicative skills over time.

This type of explorative learning is called *evaluative feedback*. In evaluative feedback there is no predefined right or wrong answer, but the environment 'responds' by assigning some local cost or utility. The learner adapts to this feedback by maximising overall utility (rewards) and minimising costs (punishments). Decision Theoretic approaches and RL both learn by evaluative feedback, where DT maximises short-term goals and RL maximises long-term goals.

Another difference between the learning techniques is that RL learns by exploration (of uncharted territory) and exploitation (of current knowledge). The ability to *explore* allows a system to learn strategies which are more robust to unseen states. The ability to *exploit* current knowledge allows learning by experience. In DT, in contrast, the dialogue cost function is learned from a fixed corpus. Thus, only state-action pairs as seen in the corpus have reliable cost values.[2]

In order to ensure enough exploration, recent research has turned to simulation-based RL, where the dialogue agent interacts with a simulated dialogue environment and thus can explore (almost) any number of strategies with (almost) no cost. Simulation-based RL has been explored by Ai and Litman (2007); Ai et al (2007b); Frampton (2008); Georgila et al (2006a); Janarthanam and Lemon (2008); Pietquin (2004); Schatzmann (2008); Thomson et al (2007); Toney (2007); Williams (2006) amongst others. We further discuss simulation-based RL in Section 3.2.3.

After we have illustrated the main advantages of RL for temporal and dynamic dialogue, we now turn to a more formal definition of the theoretical concepts of RL.

3.2 Reinforcement Learning-based Dialogue Strategy Learning

3.2.1 Dialogue as a Markov Decision Process

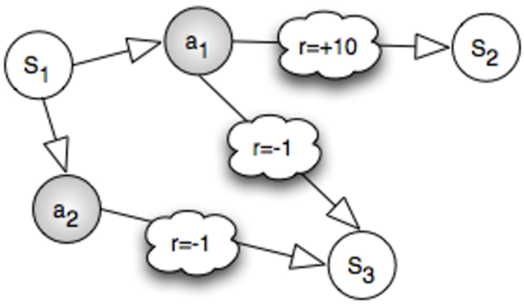

Fig. 3.2 RL with Markov Decision Processes (MDPs): The learning agent travels through a network of interconnected states. At each time t, the agent is in state s_t, takes an action a_t, transitions into state s_{t+1} according to the transition probability $p(s_{t+1}|s_t, a_t)$, and receives reward r_{t+1}

[2] Though online learning for DT has been proposed by Bohus et al (2006).

Within the RL framework for dialogue development, dialogue strategies are represented as mappings from states to actions within (MDPs), as first introduced by Levin and Pieraccini (1997); Levin et al (1998). A MDP is formally described by a finite state space S, a finite action set A, a set of transition probabilities $\mathscr{T}$ and a reward function $\mathscr{R}$. The dialogue strategy learner can be visualised as an agent travelling through a network of interconnected dialogue states (see Figure 3.2). Starting in some initial state, the learning algorithm transitions from state to state by taking actions and collecting rewards as it goes along. The transitions are non-deterministic, since the dialogue environment is stochastic and dynamic.

A MDP is a more specialised version of a (see Figure 3.3) which accounts for the temporal nature of dialogue actions: by taking an action the agent actively changes the environment and influences what states and actions are available in the consequent dialogue. The Markov Property requires that the state and reward at time $t+1$ only depend on the state and action at time t, as expressed in Equation 3.1.

$$P(s_{t+1}, r_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0) \approx P(s_{t+1}, r_{t+1} | s_t, a_t) \tag{3.1}$$

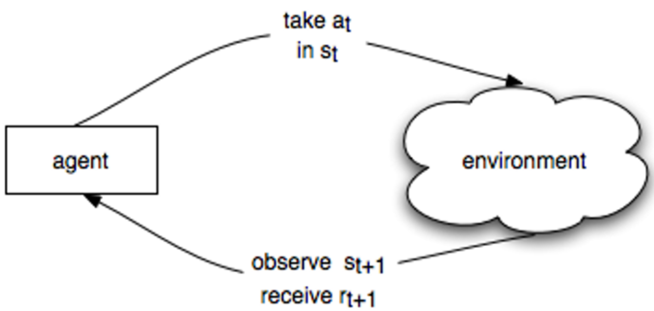

Fig. 3.3 Agent interacting with a stochastic environment and actively influencing its behaviour by taking action a_t in state s_t. The changes in the environment are observed (o_{t+1}) and a reward is received r_{t+1}

3.2.1.1 Representing Dialogue as a Markov Decision Process

We now give a short description of the tuple $\langle S, A, \mathscr{T}, \mathscr{R} \rangle$ which defines a MDP, in the case of dialogue systems.

State space (S):

The S refers to the set of reachable states for the agent within the MDP. The information represented in the state corresponds to the agent's 'view' of the environment and is usually determined by the current status of a finite number of pre-defined state variables. The state features need to meet the Markov Property, i.e. they "should summarise everything important about the sequence of positions that led to it" (Sutton and Barto, 1998). That is, the dialogue features can also represent knowledge about the dialogue history. Most systems include features of the dialogue history ($\tilde{s_d}$), together with variables representing features of the current user input action ($\tilde{a_u}$), such as the user's speech act and the confidence value returned from the speech recogniser, and task level features, also known as the features representing the "user goal" ($\tilde{s_u}$), such as filled and confirmed slots (cf. (Lemon et al, 2005; Young, 2006)), also see Figure 3.4.

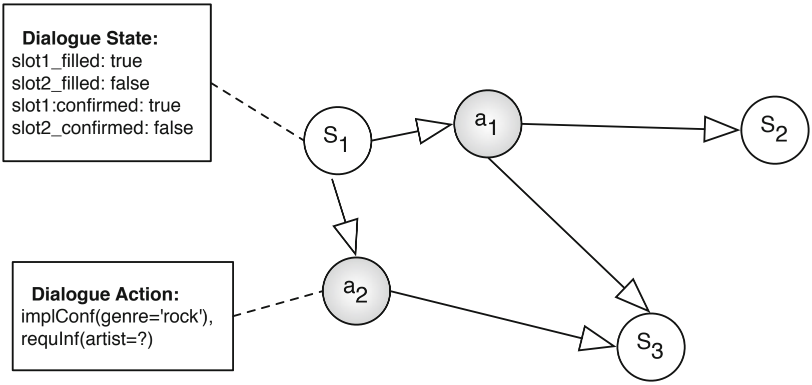

Fig. 3.4 Action space and state space for dialogue as a MDP: For slot filling dialogues the dialogue state may contain slot states; System actions can be represented as tuples consisting of dialogue act, slot name and optional slot values

Action set (A):

The action set A contains all actions available to the agent. The action set is often limited to a small number of actions, such as "request information", "implicit confirm", "explicit confirm", "present information" e.g. (Frampton and Lemon, 2006). A slightly more flexible alternative is to represent each action as a tuple consisting of speech act, slot name, and slot value, e.g. (Henderson et al, 2008), as shown in Figure 3.4. Most commonly, the dialogue actions to be learned are represented as abstract semantic Speech Acts on the intention level. For the final application the output SAs are then usually realised using template-based Natural Language generation.

State transition function ($\mathscr{T}$):

The state transition function $\mathscr{T}$ describes the dynamics of the environment, i.e. which next state $s' \in S$ is likely to follow when taking action $a \in A$ in state $s \in S$. $\mathscr{T}$ is defined over $\mathscr{T} : S \times A \times S \rightarrow [0,1]$, where

$$\mathscr{T}^a_{ss'} = P(s_{t+1} = s'|a_t = a, s_t = s); \tag{3.2}$$

Reward function ($\mathscr{R}$):

Similarly, given a current state s_t and an action a_t, the expected reward value of the next state s_{t+1} is

$$\mathscr{R}^a_{ss'} = E(r_{t+1}|s_t = s, a_t = a, s_{t+1} = s); \tag{3.3}$$

$\mathscr{R}$ plays a critical role for what is learned by the dialogue agent, as further discussed in Section 3.2.2.

3.2.1.2 Partially Observable Markov Decision Processes for Strategy Learning

A Partially Observable Markov Decision Process (POMDP) is an extension to the MDP model. POMDPs correct for the assumption that the state is fully observable, and thus provide a framework for modelling the inherent uncertainty of spoken dialogue systems (see Section 2.1). POMDPs were first introduced to dialogue strategy learning by Roy et al (2000), followed by Thomson and Young (2010); Thomson et al (2007, 2008); Williams and Young (2007b); Young (2006) more recently.

In POMDPs, the RL agent cannot directly observe the underlying state, but instead, it must make inferences about the state based on observations.[3] In contrast, a MDP approach assumes that the entire state space is fully observable. In the MDP approach uncertainty can only be represented as state features encoding ASR confidence scores, such as ‘low’, ‘high’ (cf. (Pietquin, 2006)), or whether slot is confirmed or not (confirmed slots are less uncertain than slots which are only filled). In contrast, POMDPs encode uncertainty by representing the current dialogue state s as a belief state $b(s)$ which is a distribution over the possible states. According to some current observations o this belief state gets updated, through a process called *belief monitoring*:

[3] For instance, chess is a directly observable game: you can always see the positions of all the pieces. Poker, by contrast, is partially observable: you can make inferences about the set of cards your opponents hold, but cannot directly observe them.

$$b'(s') = P(s'|o', a, b(s)) = k \times P(o'|s', a_s) \sum_{s \in S} P(s'|a, s) b(s); \quad (3.4)$$

where $b'(s')$ is the estimated belief state , $P(s'|o', a, b)$ the probability of being in a state s' given the estimated observation o', the user action a and the current belief state $b(s)$. This can be re-written as the probability of observing o' in a system state s' and given a system action a_s, given the transition probability for all the current belief states to the new state s, where k is a normalisation constant (Kaelbling et al, 1998).

Modern speech recognisers can produce a list of alternative ASR hypotheses. A POMDP-based dialogue system can then track a number of possible hypotheses at the same time, and if required, it can backtrack and correct them. Information about the user's goal can also accumulate over a series of dialogue turns. A confidence-based approach (as applied by MDPs) loses the information about alternative recognition hypotheses and has to engage in complex error recovery procedures in order to discover and correct the errors.

While being theoretically more attractive than the MDP approach, the POMDP approach faces challenges in terms of scaling up to real-world problem domains. Many POMDP approaches are only applicable to simple form-filling problems using only a small number of states, such as pizza ordering (Williams, 2006). The state spaces for POMDPs often explicitly represent all the feature values (in order to update the belief state), in contrast to MDPs where slot values can be approximated. As a result, belief monitoring for POMDPs is computationally very expensive and is often intractable for practical spoken dialog systems. How to scale-up POMDPs to real world problems is a research topic in its own right, and recent progress has been made, e.g. (Crook and Lemon, 2010, 2011a; Lemon et al, 2008; Thomson and Young, 2010; Thomson et al, 2007, 2008; Williams and Young, 2007b).

This book explores the MDP framework for dialogue strategy learning, which is frequently used for practical systems, e.g. (Heeman, 2007; Henderson et al, 2005, 2008; Janarthanam and Lemon, 2010c; Levin et al, 2000; Pietquin and Dutoit, 2006b; Rieser et al, 2010; Singh et al, 2002; Spitters et al, 2007; Walker, 2000) etc.

3.2.2 The Reinforcement Learning Problem

The MDP model allows a dialogue management strategy (or policy) $\pi : S \rightarrow A$ to be viewed as a mapping from states to actions: For every state s, the policy selects the next system action a. The MDP model also enables us to formalize dialogue management as a mathematical optimisation problem which can be solved using Reinforcement Learning (Sutton and Barto, 1998).

3.2.2.1 Elements of Reinforcement Learning

The optimal policy π^* is the policy that that selects those actions that yield the highest final reward over the course of the dialogue. We can define the final reward $\mathscr{R}$ to be the total discounted return that the learning dialogue manager receives when starting at time t (see Equation 3.5). A discount factor γ, with $0 \leq \gamma \leq 1$, can be used to weight immediate rewards more strongly than distant future rewards. For $\gamma = 0$, RL only maximises the immediate utility of an action, similar to the previously discussed Decision Theoretic approaches. For dialogue strategy learning γ is commonly set close to 1 to take future rewards into account as strongly as possible.

$$\mathscr{R}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{3.5}$$

The (estimated) reward received at time step t indicates the intrinsic desirability of a state or action. The optimal policy π^* is discovered via trial-and-error search through interaction between the learning agent and its dynamic environment. Thereby the agent learns a *value function* $V^\pi(s)$ which reflects the long-term desirability of a state by taking into account all the states that are likely to follow it. The associated V-value of a state s expresses how much future reward can be expected when visiting state s and subsequently following policy π (see Equation 3.6).

$$V^\pi(s) = E_\pi(\mathscr{R}|s_t = s) \tag{3.6}$$

The value function can be re-written as the Q-function $Q^\pi(s,a)$ which estimates the expected return of taking action a in a given state s and following the policy π thereafter (see Equation 3.7).

$$Q^\pi(s,a) = E_\pi(\mathscr{R}|s_t = s, a_t = a) \tag{3.7}$$

$V^\pi(s)$ and $Q^\pi(s,a)$ can be formulated recursively, given the tight correlation of a state value to the value of its potential successor states, as expressed in the transition function $\mathscr{T}^a_{ss'}$ (see Equation 3.2) and reward function $\mathscr{R}^a_{ss'}$ (see Equation 3.3). The resulting Equations 3.10 and 3.11 are also called the *Bellman equations* (Bellman, 1957):

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathscr{T}^a_{ss'} [\mathscr{R}^a_{ss'} + \gamma V^\pi(s')] \tag{3.8}$$

$$Q^\pi(s,a) = \sum_{s'} \mathscr{T}^a_{ss'} [\mathscr{R}^a_{ss'} + \gamma V^\pi(s')] \tag{3.9}$$

where $\sum_a$ represents the summation over available actions in state s, and $\sum_{s'}$ the summation over potential successor states of s. The Bellman equations describe the expected reward for taking the action prescribed by some policy π. The equations for the optimal policy π^* are referred to as the *Bellman optimality equations* :

$$V^*(s) = \max_a \sum_{s'} \mathscr{T}^a_{ss'}[\mathscr{R}^a_{ss'} + \gamma V^*(s')] \quad (3.10)$$

$$Q^*(s,a) = \sum_{s'} \mathscr{T}^a_{ss'}[\mathscr{R}^a_{ss'} + \gamma \max_a Q^*(s',a')] \quad (3.11)$$

Finding an optimal policy by solving the Bellman Optimality Equation requires accurate knowledge of the environment dynamics, a substantial amount of time and space to do the computation, and the Markov Property (Sutton and Barto, 1998). Since these assumptions can hardly be met by any real-world application, a number of different algorithms are available to find approximate solutions, as described in the next Section.

3.2.2.2 Algorithms for Reinforcement Learning

Sutton and Barto (1998) distinguish three elementary classes of algorithms to solve Reinforcement Learning problems: Dynamic Programming (DP), Temporal Difference (TD) learning, and Monte Carlo methods. DP is a "model-based" approach to RL, whereas TD and Monte Carlo are both model-free (also known as simulation-based) approaches. Model-based approaches explicitly model the dynamics of the environment to compute an estimate of the expected value of each action. We further explain and discuss model-based and simulation-based RL approaches to dialogue strategy learning in Section 3.2.3.

In the following we first explain the general mechanisms underlying all RL algorithms. We then compare DP with TD in order to highlight the general differences between model-based vs. model-free/simulation-based algorithms. We then introduce the two most popular forms of TD learning: the SARSA algorithm and Q-learning.

Most RL algorithms are based on the same basic mechanism: they learn by incrementally updating the expected Q-values for each state action pair, estimating the Bellman optimality equation. They start with an initial value function, where all the state-action values are initialised to some arbitrary value. We can visualise the learning process as a matrix of states versus actions, where the cells are the Q-values, as in Figure 3.1.

	s_1	s_2	s_3	s_4
a_1	0.0	0.0	0.0	0.0
a_2	0.0	0.0	0.0	0.0
a_3	0.0	0.0	0.0	0.0

Table 3.1 Initialised state-action values before learning for a learning problem with 4 states s_{1-4} and 3 actions a_{1-3}

The expected Q-value for a state-action pair Q_k is updated in each iteration k, taking the general form

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate] \tag{3.12}$$

where the step-size parameter describes the effect-size of changes from time step t to time step $t+1$. Step size is also called the *learning rate* (α). $[Target - OldEstimate]$ is the error towards the optimal value Q^*. The optimal Q-value is guaranteed if $Q_{t\to\infty} = Q^*$. In practise, the learning algorithm stops if Q_k is believed to be sufficiently close to Q^*. One stopping criterion is to observe whether the Q-values converge, i.e. when the difference $|Q_t(s,a) - Q_{t-1}(s,a)|$ is lower than some threshold.

The learned Q-values after strategy training in our example might then be as in Table 3.2. The optimal policy π^* selects the system action a with the highest expected value in each state s.

	s_1	s_2	s_3	s_4
a_1	2.40	1.80	**4.05**	0.46
a_2	**3.67**	1.38	2.01	**2.78**
a_3	0.9	**2.90**	2.52	1.24

Table 3.2 Learned state-action values: The optimal strategy selects the action a with the highest expected value in each state s

The essential difference between Dynamic Programming and Temporal Difference Learning is in the way that they update Q-values. DP works by updating the Q-value *off-line* for *every* possible state-action pair in a single iteration. It therefore requires an explicit model of the dynamics of the environment, where the transition function fully defines the probability of moving from any state s to another state s', i.e. the state transition probability $P(s'|s,a)$, and the reward function $R(s,a)$ defines the reward for choosing the action that creates this transition (see Equation 3.13).

$$Q_t(s,a) \leftarrow R(s,a) + \sum_{s'} P(s'|s,a) \max_{a} Q_t(s',a') \tag{3.13}$$

Temporal Difference learning, in contrast, does not require a full model of the transition function to be available, i.e. it is *model-free*. TD only requires some sample episodes of state-action transitions, instead of considering all possible transitions. Only the sampled transitions contribute to the improved estimate of Q^* (see Equation 3.14). TD therefore requires the *online* exploration of sufficiently large number of state-action pairs in order to reduce the error.

$$Q_t(s,a) \leftarrow Q_t(s,a) + \alpha\,[\underbrace{R(s,a) + Q_t(s',a')}_{Target} - \underbrace{Q_t(s,a)}_{OldEstimate}\,] \tag{3.14}$$

Temporal Difference learning can be implemented as an *on-policy* algorithm called SARSA (Rummery and Niranjan, 1994), and also as an *off-policy* algorithm called Q-learning (Watkins and Dayan, 1992). On-policy learning updates the policy based on actions taken by the agent. Off-policy learning also can learn about policies other than that currently followed by the agent. Both approaches have their advantages and disadvantages, see (Sutton and Barto, 1998) for a discussion.

Previous work has applied a variety algorithms for learning dialogue strategies. (English and Heeman, 2005; Levin et al, 2000), for example, use Monte Carlo methods, while Singh et al (2002) use Dynamic Programming, and Schatzmann et al (2005b); Walker (2000) apply Q-learning. Frampton and Lemon (2006); Henderson et al (2008); Janarthanam and Lemon (2010c); Rieser et al (2010) use the SARSA(λ) algorithm. Currently, (different versions of) SARSA are used most widely as they are easy to implement and are known to converge rapidly.

In this book we will use a RL method which implements different versions of the SARSA(λ) algorithm, where the λ refers to the use of an eligibility trace . An eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event so that the current reward can be attributed to past states.

The name SARSA reflects the fact that the main function for updating the Q-value depends on the current state of the agent $\mathbf{s}_t$, the action that the agent choses $\mathbf{a}_t$, the reward $\mathbf{r}_{t+1}$ that the agent gets for choosing this action, the state $\mathbf{s}_{t+1}$ that the agent will be in after taking that action, and finally the next action $\mathbf{a}_{t+1}$ that the agent will choose in its new state, as shown in Algorithm 1. The degree of "greediness" defines the exploration-exploitation trade-off for online policy training. By using an ε-greedy policy, the so far learned optimal action is taken with probability $(1-\varepsilon)$ while exploration moves are executed with probability ε.

Algorithm 1 *SARSA*

$Q(s,a) \Leftarrow$ arbitrarily

repeat {for each episode:}
  Initialise s
  Choose a from s using policy derived from Q (e.g. ε-greedy)
  for all steps in the episode **do**
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g. ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'; a \leftarrow a'$;
  end for
until s is terminal

3.2.2.3 The Curse of Dimensionality, and State Space Reduction

In Reinforcement Learning the number of possible policies grows exponentially with the state and action spaces ($no.\ policies = no.\ actions^{no.\ states}$ and $no.\ states = no.\ values^{no.\ features}$). For example, if the state space consists of only 4 binary features, and only 3 actions are available for learning, the total number of policies is already as high as $3^{2^4} = 43,046,721$. RL algorithms rely on exhaustive trial-and-error search, which makes learning with large policy spaces computationally very expensive. For example, Henderson et al (2008) report that with random trial-and-error search to learn a complex problem, the algorithm would have visited only one policy in every $74^{10^{385.6}}$ after 1 year, if they would have been able to explore policies at a rate of 1 policy a second.

On the other hand, the richer the state space, the more expressive the system and the policy which can be learnt. This dilemma for strategy learning is also known as the “curse of dimensionality” (Bellman, 1957).

In general, there are two different methods to overcome this problem: either to reduce the state space or facilitate cheaper training techniques (such as simulation-based learning), also see Section 2.3.4. In this Section we review different techniques for state-space reduction, and we discuss simulation-based methods in Section 3.2.3.

In order to reduce the state space for learning, previous research has investigated different techniques, such as reducing the number of features, reducing the number of possible state-action combinations by structural restrictions (or so-called “sub-policy learning”), or summarising states which are similar.

Work by Frampton and Lemon (2006); Tetreault and Litman (2006), for example, reduces the state space by identifying the most ‘valuable’ state space features for learning, using feature-selection methods. They identify the value of a state feature by measuring the increase in reward gained (Frampton and Lemon, 2006), or training time taken until strategies converge (Tetreault and Litman, 2006).

Work by Cuayáhuitl et al (2006); Heeman (2007); Singh et al (2002) applies sub-strategy learning: Singh et al (2002) hand-code a policy and leave difficult design decisions for optimisation, while Cuayáhuitl et al (2006) apply hierarchical Reinforcement Learning, and Heeman (2007) learns with hand-coded constraints. Similarly Williams (2008) uses a conventional dialogue manager to constrain the choices available for optimisation by a POMDP system.

Another line of research groups together state spaces which are similar (according to some definition of similarity) into one single state, e.g. (Crook and Lemon, 2011a; Denecke et al, 2004; Goddeau and Pineau, 2000; Henderson et al, 2008; Williams and Young, 2007b). For example, recent work using Value-Directed Compression automates the discovery of states which have similar values and can therefore be treated as equivalent (Crook and Lemon, 2011a). Henderson et al (2008) combine RL with linear function approximation, which we now explain in more detail.

In order to learn with linear function approximation the dialogue state is represented as a vector of real valued features $f(s)$. For each state s the vector $f(s)$ is mapped to a vector of estimates of $Q(s,a)$, one estimate for each action a. Given weights trained on data, the Q-function can be re-written as the inner product of state vector $f(s)$ and the weighted vector w_a:

$$Q^{\pi}(s,a) = f(s)^T w_a = \sum_t f(s) w_{ai} \tag{3.15}$$

This approximation method has the effect that two states are treated as similar if they share features. During training, the update function will affect states which share feature i. Each feature represents a dimension with respect to which two states can be similar or different. This similarity measure is known as a linear kernel.

In this book we will experiment with several of these techniques. First, we select the most promising features for state space design. (Section 7.4.3.2) We also formalise the learning problem as a hierarchy of decisions and apply hierarchical Reinforcement Learning (Section 7.5.3), we limit the action choices to be learned by encoding domain knowledge into Information State Update rules (Section 7.5.4), and we also apply linear function approximation (Section 7.11.1).

Most importantly, we facilitate "cheap" learning by training our policy in a simulated learning environment, also known as simulation-based RL. We combine this method with another dialogue simulation technique known as a "Wizard-of-Oz" methodology. We further explain these techniques below.

3.2.3 Model-based vs. Simulation-based Strategy Learning

In general, there are two different approaches to dialogue strategy learning: model-based and simulation-based (also known as 'model-free') RL. In Section 3.2.2.2 the corresponding algorithms were presented. This Section now summarises and discusses their respective advantages and disadvantages for dialogue strategy learning.

3.2.3.1 Model-based Reinforcement Learning

Model-based approaches explicitly represent a model of the dynamics of the environment to compute an estimate of the expected value of each action (see Figure 3.5). Learning is then done *off-line* without any interaction between the learning agent and the environment using Dynamic Programming techniques. The model-based approach neglects the actual user responses to system actions – it only considers the system-state transitions and the rewards associated with them. The transition probabilities $\mathscr{T}^a_{ss'}$ are estimated from a corpus in which the state transitions have been logged. Parameter estimation can be done using simple Maximum Likelihood Estimation based on the relative frequency of occurrence of each transition. For off-line learning the final reward is already present in the corpus, i.e. it is known in advance whether the strategy is successful or how it is rated by the user. As such, $\mathscr{R}^a_{ss'}$ can be directly estimated as well. With a model, the agent can reduce the number of steps to learn a policy by simulating the effects of its actions at various states. For model-based approaches it is possible to derive a policy that is guaranteed to be optimal *with respect to the data*.

However, learning from fixed data sets has a number of significant deficiencies:

- Currently available corpora are not large enough to reliably estimate all transition probabilities for practical systems (Lemon and Pietquin, 2007).

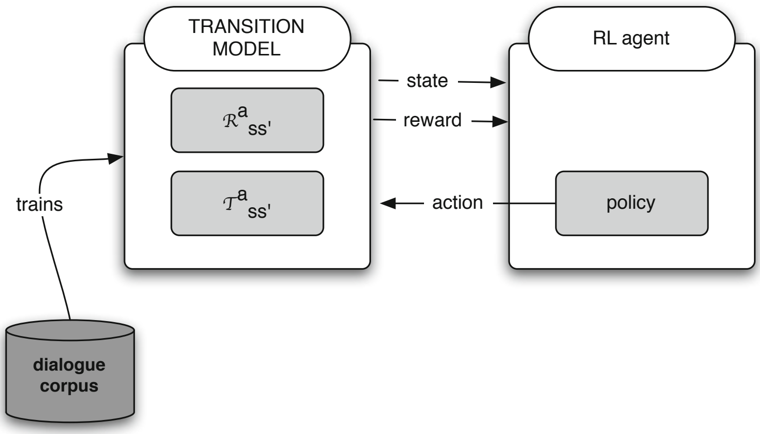

Fig. 3.5 Model-based Reinforcement Learning: Learning a transition model for the transition function $\mathscr{T}^a_{ss'}$ and the reward function $\mathscr{R}^a_{ss'}$ from data

- When learning from a fixed corpus, the dialogue strategy can only use state-action combinations that were explored at the time of the corpus data collection. It cannot explore entirely new strategies, since no transition probabilities can be computed for unseen state-action combinations. A deviation in user behaviour during the live operation of the system is likely to move the system into an unexplored portion of the state space, possibly causing the system to fail. Furthermore, the optimal strategy is not guaranteed to be present in the training corpus.
- Learning from fixed data sets has the principal drawback that learning is limited to domains where working systems already exist. The actions for learning are pre-defined by the system capabilities. The state space also needs to be known in advance so that the corpus can be annotated accordingly, and/or features can be logged during the data collection. If the chosen state space representation or action set turns out to be problematic, it is difficult to change afterwards.

3.2.3.2 Simulation-based Reinforcement Learning

Simulation-based (also known as 'model-free') RL approaches do not explicitly represent the dynamics of the environment, but instead directly approximate a value function. This is done *online* by interaction with a simulated environment (see Figure 3.6). Simulation-based learning involves two stages: First, different simulated components of the dialogue environment (such as a simulated user and an error model) are trained on a given corpus using Supervised Learning techniques. The dialogue strategy is then trained by simulated interaction using Monte Carlo or Temporal Difference learning techniques (see previous Section 3.2.2.2). These approaches offer near-optimal solutions that depend on systematic *exploration* of all

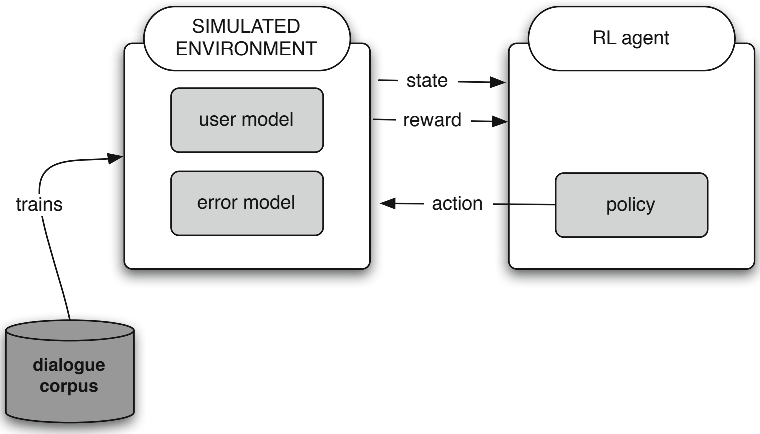

Fig. 3.6 Simulation-based Reinforcement Learning: Learning a stochastic simulated dialogue environment from data

actions in all states. Hence, the simulated components need to reliably generalise to unseen dialogue states in order to support this exploration. As such, simulation-based RL is a more complex approach than directly learning from a fixed data set, but it offers significant advantages:

- The simulated user/environment allows any number of training episodes to be generated, so that the learning agent can exhaustively explore the space of possible strategies.
- It enables strategies to be explored which are not in the training data. The learner can deviate from the known strategies and experiment with new and potentially better strategies.
- The system state space and action set do not need to be fixed in advance, because the system is not directly trained on corpus data. If the given representation turns out to be problematic, it can be changed and the system retrained using the simulated user.

Simulation-based RL, however, also faces challenges:

- The quality of the learned strategy depends on the quality of the simulated environment. Hence, appropriate methods to evaluate the simulated components are necessary.
- The reward signal cannot be read off from the data, but the reward function has to be explicitly constructed.
- Results obtained in simulation may not be an accurate indication of how the strategy would perform with real users (though see results by e.g. (Janarthanam et al, 2011; Lemon et al, 2006a)).
- The simulated components need to be trained on in-domain data, which is expensive to collect. In cases for new application domains where a system is designed

from scratch, however, there is often no suitable in-domain data available. Collecting dialogue data without a working prototype is problematic, leaving the developer with a classic chicken-and-egg problem.

In the course of this book we will address these challenges by introducing new methods and techniques. In particular, we propose 'bootstrapping' a simulated learning environment from Wizard-of-Oz data, as outlined in Chapter 5. In the next Section we explain more about simulated interaction and Wizard-of-Oz studies.

3.3 Dialogue Simulation

We now explain some general background on simulating dialogue. A dialogue simulation approximates real Human-Computer Interaction (HCI) in order to facilitate dialogue system development and testing. Two main simulation techniques exist: the so-called 'Wizard-of-Oz' experiment and computer-based simulation.

3.3.1 Wizard-of-Oz Studies

Wizard-of-Oz (WOZ) studies are performed right at the beginning of the design process, before a working prototype is available (Dahlbäck et al, 1993; Fraser and Gilbert, 1991). In a WOZ experiment, a hidden operator, the 'wizard', simulates some aspects of the behaviour of the application, while subjects are led to believe that they are interacting with a real system. Hence, the wizard's speech is rendered using synthetic speech. It is important to keep up the illusion of real HCI, as users behave differently towards other people than towards machines, for example, they use simpler language, shorter utterances, and different pragmatic behaviour, see (Doran et al, 2001; Jönsson and Dahlbäck, 1988; Krause and Hitzenberger, 1992; Moore and Morris, 1992).

At the end of the interaction the user is asked to fill out a questionnaire or otherwise rate the interaction (see Figure 3.7). The main purpose of a WOZ experiment is to gather initial insights about user behaviour and user preferences. Furthermore, the user utterances are recorded and may be used to develop a language model and/or acoustic model for the prototype system.

A central difference between WOZ studies and real HCI is that the wizard usually has perfect recognition and full understanding of the user's utterances, whereas for real HCI the interaction is affected by noise as introduced by ASR and NLU. Hence, recent work has suggested introducing artificial noise in the WOZ setup in order to estimate real HCI more closely, e.g. (Passonneau et al, 2011; Rieser et al, 2005; Schlangen and Fernandez, 2007; Skantze, 2005; Williams and Young, 2004a). In Chapter 6 we will describe a WOZ study where we use a similar setup.

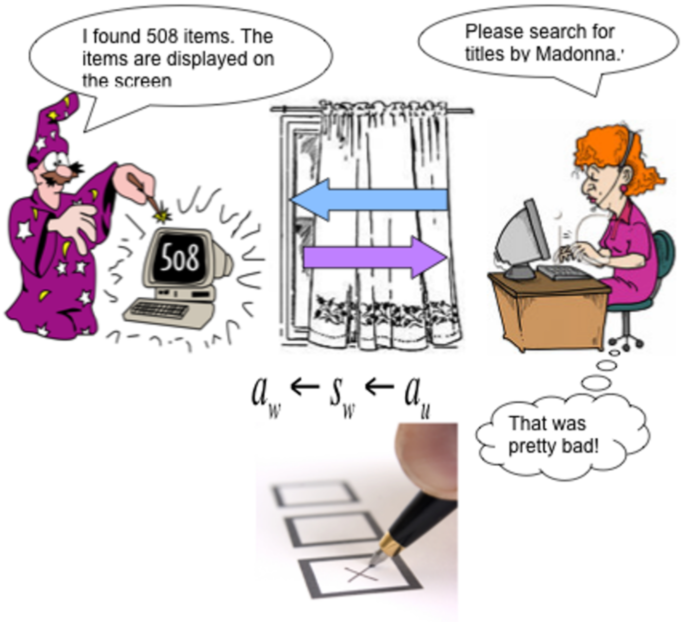

Fig. 3.7 Wizard-of-Oz simulation setup: Hidden operator (the "wizard") simulates dialogue behaviour; user interacts in the belief that they are talking to a machine, and rates the dialogue behaviour

3.3.2 Computer-based Simulations

In a computer-based simulation a prototype automated system interacts with a simulated user (see Figure 3.8). Computer-based simulations are used for different purposes: testing and debugging prototype systems, e.g. (Chung, 2004a; Engelbrecht et al, 2009; López-Cózar et al, 2003), as well as for automatic strategy development, for example using Supervised Learning, e.g. (Filisko and Seneff, 2006), or Reinforcement Learning, e.g. (Schatzmann et al, 2006). A more detailed classification can be found in Chapter 7. In the following we focus on simulated environments for RL, and summarise some of their general characteristics.

In simulation-based RL, the RL agent explores the expected return value for different strategies while interacting with a simulated environment. The interaction usually takes place on the intention level, where system and simulated user exchange speech acts (a_u, a_s). The simulated components of the dialogue environment are commonly statistical models obtained via Supervised Learning from dialogue corpus data. An error model introduces errors in order to simulate error-prone ASR.

How close the simulated interaction is able to resemble real HCI in general, depends on the quality of the simulated components. Various evaluation measures for quality assurance have been proposed, which we will review in Chapter 7.1.1. An-

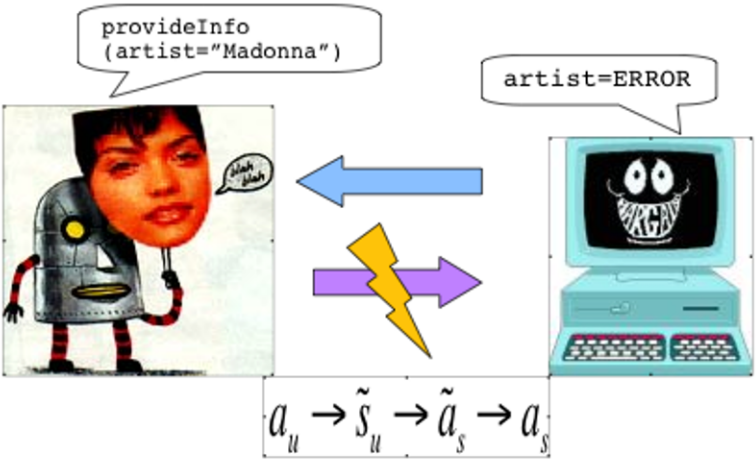

Fig. 3.8 Computer-based simulation setup: The dialogue manager (right) interacts with a simulated user (left) over a noisy channel

other difference to real HCI is that simulated users cannot rate the system according to their personal preferences. In simulation-based RL, the strategy is rated by the reward function, which is specified by the system designer. It is an open research question whether real users will "reward" the real dialogues accordingly (Paek, 2006). In addition, how to qualitatively and quantitatively measure the differences between simulated and real dialogues is on ongoing research issue (Ai and Litman, 2006; Williams, 2007). We will address both questions in Chapter 8.5.

3.3.3 Discussion

This book proposes to combine the two simulation methods into one framework: we learn the simulated environment for RL from data collected in a WOZ experiment. This approach enables automatic strategy learning in domains where no prior system is available. Optimised learned strategies are then available from the first moment of online-operation, and handcrafting of dialogue strategies is avoided. This independence from the availability of in-domain dialogue data allows researchers to apply RL to completely new application areas beyond the scope of existing dialogue systems. We call this method 'bootstrapping'. In the next Section we present the application domain where we will implement our method.

3.4 Application Domains

3.4.1 Information-Seeking Dialogue Systems

Reinforcement Learning has been applied to many different domains, from simple slot-filling dialogues, such as pizza ordering (Williams, 2006), to complex tutoring tasks (Tetreault and Litman, 2006). In this book we apply RL to optimise *information-seeking* dialogue strategies, also known as "information retrieval" dialogues. Information-seeking dialogues serve as an interactive natural language interface to database search (Androutsopoulos and Ritchie, 2000). A varying number of different search candidates is retrieved, dependent on the provided constraints and the size of the database. Some search constraints may also be left unspecified. *Form-filling* dialogues are similar to information-seeking dialogues, as they follow the basic structure of information acquisition and presentation following each other. However, form-filling dialogues are also different, as most of the task constraints are compulsory. (Note that there is also the option for the user to provide the constraint "I don't care" – as long as the form gets filled.) For example, the user always has to specify date, time, origin and destination city in order to book a flight. Thus, the decision of when to start the information presentation phase is trivial for form-filling dialogues, as the database only gets queried when the user has provided all the necessary information, i.e. at the very end of the information acquisition phase, e.g. (Lemon et al, 2006b).

In contrast, information seeking dialogue strategies can be applied in domains where information can be left open/unspecified without causing the task to fail. For example, digital music and library catalogue search, searching for a restaurant, searching the web, or "interactive question answering" tasks (Webb and Webber, 2008). Thus, one major decision for information-seeking dialogue strategies is *when* to stop asking for further constraints and to present the retrieved information to the user. This question has also been addressed in the context of RL, e.g. (Heeman, 2007; Levin et al, 2000; Pietquin, 2006), as we further discuss in Section 4.1.

We distinguish between an *information acquisition* and an *information presentation* phase, which we define as follows:

- *Information acquisition*: during this initial phase the system is gathering information from the user. During this phase, the high-level communicative goals that the system tries to achieve are are fairly complex: the goals include getting the user to supply information, and explicitly or implicitly confirm information that the user has supplied. Here, one major decision for information-seeking dialogue systems is *when* to stop asking for more constraints.
- *Information presentation*: this second phase commences when the system has obtained information that matches the user's requirements and the options (e.g. flights, restaurants, or songs) need to be presented to the user. Here, the communicative goal is mainly to convey a certain set of facts to the user, possibly in conjunction with a request for a choice among these options. Here, one major

decision for information-seeking dialogue systems is *how* to present the information.

Note that Bangalore et al (2001) also distinguish these two phases, where they define the first phase to be traditionally associated with Dialogue Management, and the second phase to be a traditional Natural Language Generation task. According to (Bangalore et al, 2001) one of the major current challenges is to combine both in a unified framework. We present such a framework in this book. In Chapter 7 jointly optimise *when* to start the information presentation phase (a task which is traditionally associated with DM) and then decide on the presentation medium (multimodal versus speech only). In Chapter 9 we jointly optimise information presentation strategy and attribute selection, which are both higher level NLG tasks.

Another major challenge for information-seeking dialogue systems is how to deal with bad ASR quality. As a voice interface to large databases, information-seeking dialogue systems are a challenging task for automatic speech recognition, as a large set of proper nouns needs to be recognised. Other research investigates different solutions to this problem, e.g. (Filisko and Seneff, 2005; Gruenstein and Seneff, 2006, 2007). However, the question of how to improve ASR quality for large databases is beyond the scope of this book .

3.4.2 Multimodal Output Planning and Information Presentation

The term "multimodal" has several different meanings (Rudnicky, 2005). In this book we address the problem of multimodal output planning, where two parallel modes are available for information presentation: one using speech only (`verbal`) and one showing a list of retrieved items on the screen while implicitly confirming using speech (`MM`), as in Example 3.1.

Example 3.1.
`verbal` : "For $[constraint1]$ and $[constraint2]$ I found $[n]$ items in the database: $[i_1], [i_2], \ldots [i_n]$. Which one would you like?"

`MM` : "For $[constraint1]$ I found $[n]$ items in the database. A list of possible candidates is displayed." (*see example in Figure 3.9*)

iTALK Options Viewer

SELECTED	ARTIST	TITLE	ALBUM	GENRE
☐	Eric Clapton	Rolling And Tumblin	MTV Unplugged	Blues
☐	Eric Clapton	Old Love	MTV Unplugged	Blues
☐	Eric Clapton	Malted Milk	MTV Unplugged	Blues
☐	Eric Clapton	San Francisco Bay Blues	MTV Unplugged	Blues
☐	Eric Clapton	Alberta	MTV Unplugged	Blues
☐	Eric Clapton	Walking Blues	MTV Unplugged	Blues
☐	Eric Clapton	Running on Faith	MTV Unplugged	Blues
☑	Eric Clapton	Layla	MTV Unplugged	Blues
☐	Eric Clapton	Nobody Knows You When You ...	MTV Unplugged	Blues
☐	Eric Clapton	Lonely Stranger	MTV Unplugged	Blues
☐	Eric Clapton	Tears in Heaven	MTV Unplugged	Blues
☐	Eric Clapton	Hey Hey	MTV Unplugged	Blues
☐	Eric Clapton	Before You Accuse Me	MTV Unplugged	Blues
☐	Eric Clapton	Sign	MTV Unplugged	Blues

Fig. 3.9 Example of a list of retrieved items being displayed on the screen

The user input can also be multimodal: the user can speak and make selections on the screen. Nevertheless, our main focus is on multimodal output planning. There are two main questions for multimodal output planning: *when* to use *which* medium, and how to *distribute* information across media (also known as “media allocation” or “multimodal fission”). This book concentrates on the former question.

An issue with multimodal interfaces is, however, the fact that multimodal *output* is not always appropriate in every context, and needs to be adapted to the “cognitive load” of the user (Oviatt, 2006; Oviatt et al, 2004). Dialogue systems typically are used in so-called “eyes-and-hands-busy” situations, i.e. where the user performs another task in parallel which requires his visual attention, such as driving for in-car applications or homework for in-home applications. To then display information on a screen can only be beneficial to a certain extent. The appropriate choice of medium when the user is performing a eyes-busy task depends on various factors, e.g. the attention the user spends on the primary task (also known as the “cognitive load” (Sweller, 1988)), the amount of information to be presented, screen size, user preferences, etc. As such, multimodal output planning is a strong candidate for machine learning techniques (Rudnicky, 2005), as it requires context-dependent and adaptable strategies. To the authors’ knowledge this book is the first to apply RL to the multimodal output planning problem.

Chapter 9 investigates more sophisticated Natural Language Generation (NLG) techniques for information presentation. Different content-planning techniques for information presentation are investigated, such as summaries tailored to the user type (Demberg and Moore, 2006; Walker et al, 2004a,b), summaries based on the expected information gain with respect to the database structure (Chung, 2004a; Polifroni and Walker, 2006; Polifroni et al, 2003), data-driven content planning for multimodal output generation (Guo and Stent, 2005), as well as data-driven methods for lower-level aspects, such as sentence planning (Stent et al, 2004a) or word-choice (Stent et al, 2008). So far, however Dialogue Management (DM) and NLG have always been considered to be separate tasks, which are performed in a pipeline: first the DM determines *what* to say and then the NLG component determines *how* to say it. We argue that DM and NLG are two closely interrelated problems for information seeking dialogues: the decision of *when* to present information depends on the available options for *how* to present them, and vice versa. In this book we take a first step towards an integrated model of DM and NLG: we formulate the problem as a hierarchy of joint learning decisions which are optimised together (Lemon, 2011). This hierarchical approach to DM and NLG is now being explored by other researchers (Dethlefs and Cuayahuitl, 2010; Dethlefs and Cuayáhuitl, 2011). The central NLG task we address in the following chapters is to determine the output medium. However, we will also address other NLG tasks such as learning of Information Presentation policies later, in Chapter 9.

3.4.3 Multimodal Dialogue Systems for In-Car Digital Music Players

This book applies RL to optimise information-seeking strategies for multimodal dialogue systems which serve as an interface to a in-car digital music player. This application domain offers interesting aspects for learning. The in-car scenario is a eyes-and-hands-busy environment, where strictly multimodal solutions may not always be appropriate. Results of a study on multimodal in-car interaction by Salmen (2002) show that the timing of multimodal presentation is important for the driving performance, where the main focus should be on speech-based output and input. On the other hand, a study by Kun et al (2007) suggests that verbal interaction leads to decreased driving performance for conditions with low ASR quality. Furthermore, a study by Hu et al (2007); Winterboer et al (2007) indicates that strategies are perceived differently, dependent on the dialogue environment: while for driving situations with low cognitive load users prefer strategies which are most effective, these are not preferred in situations with high cognitive load. In sum, media choice depends on various contextual and environmental features when the user is driving. Hence, learning strategies which automatically adapt to various contextual features is a promising direction of research.

3.5 Summary

We argued that RL is well-suited for dialogue strategy development, as dialogue is learned by evaluative feedback with delayed rewards and exploration which suit the temporal and dynamic nature of dialogue. We then provided further background on RL and how it is applied to dialogue strategy learning, where we highlighted the advantages of simulation-based RL. We presented two different simulation techniques for dialogue development: Wizard-of-Oz studies and computer-based simulation, and we proposed to combine both into one framework in order to reduce development costs even further. Finally, we introduced the concepts of information-seeking dialogue systems and multi-modal output planning in the application domain, where we discuss several prototype systems which serve as an interface to a digital music player for the in-car domain. We proposed to replace the current hand-coded threshold-based approaches with automatic RL techniques.

In the next Chapter we provide a“proof-of-concept” that RL outperforms manual strategy design for a wide range of application scenarios.

Chapter 4
Proof-of-Concept: Information Seeking Strategies

Fig. 4.1 Galileo's Telescope - a proof-of-concept Information Seeking device

4.1 Introduction

In the following Chapter we provide a "proof-of-concept" that RL outperforms hand-coded strategies which are manually tuned to the same reward function. We also show this for a wide range of application scenarios. We use simulation-based optimisation, i.e. we adapt different strategies to different simulated environments. We also explain and illustrate the major practical advantages of RL policies over hand-coded heuristics and provide a description of the qualitative differences. This proof-of-concept study is necessary to motivate the data-driven "bootstrapping" approach that we introduce in Chapter 5.

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_4,

A number of previous studies compare RL-based strategies against hand-coded rule-based strategies, e.g. (Frampton and Lemon, 2006; Heeman, 2007; Lemon and Liu, 2007; Lemon et al, 2006a; Scheffler and Young, 2002; Spitters et al, 2007; Walker, 2000). Their results indicate that RL strategy learning in general outperforms hand-coding. However, this method of comparison has been criticised as unfair, since the hand-coded strategies are not tuned to the same reward function (Paek, 2006).

4.1.1 A Proof-of-Concept Study

For our proof-of-concept study we make the following assumptions and simplifications. First of all, we simplify the learning task. In the long run we aim to learn information-seeking dialogue strategies including the choice of output medium (see Section 3.4.2). For the proof-of-concept, however, we only learn the decision of *when* to end the 'information acquisition' phase and to commence with 'information presentation' (as defined in Section 3.4.1). For the proof-of-concept study we do not learn *how* to present (e.g. which medium to choose), but we always present items multimodally, i.e. displaying them on the screen while naming the number of retrieved items verbally. After the information is presented to the user we assume that the user selects one of the displayed items and the dialogue ends. Later in the book we will investigate different information presentation methods.

Example 4.1 illustrates this approach (where "db:" denotes the number of returned database items). Note that simulation-based strategy development takes place on the intention level, as explained in Section 3.3. For the purpose of clarity we choose to illustrate the simulated dialogues with examples from the MP3 music domain. However, we do not aim to model a specific type of domain, but rather, we want our proof-of-concept study to be generic for different applications of information seeking dialogue (see Section 3.4.1).

Example 4.1.

```
System: ``Hello, how may I help you?''
User:   ``I want a Radiohead song''
System: ``Ok, a Radiohead song. From what album?'' (db: 77)
User:   ``From OK Computer''
System: ``OK from OK Computer. What song title?''(db: 12)
User:   ``Paranoid Android''
System: ``Did you say Karma Police?'' (db: 1)
User:   ``No''
System: ``OK, there are 12 items you might be looking for,
          which are now shown on the screen.'' (db:12)
```

It is also desirable for our proof-of-concept study to cover a wide spectrum of application scenarios. We therefore manually construct various simulated environments to span a wide range of possible applications, as we will explain in the rest of this chapter. Our objective is to show that our policy learning framework covers this spectrum.

Prior work on RL for information seeking dialogue strategies has used two implicit thresholds for representing the number of retrieved database items ("DB hits"); one for representing the number of DB hits in the state space by quantising them (low-medium-high), and one in the reward function (Heeman, 2007; Levin et al, 2000; Pietquin, 2006) (cf. discussion in Section 7.10). Policy learning is restricted by these various thresholds, since it only can operate on a quantised state space and no fine-grained decisions can be learned.

The vast majority of dialogue policy learning research does not consider the *number* of database results at all, resulting in policies which are not sensitive to the current number of search results (e.g. (Henderson et al, 2005, 2008; Singh et al, 2002)). This is obviously bad: if there is only one DB search result for a given set of user constraints, we should tell it to the user immediately, regardless of how many information search slots have been filled in the dialogue. Conversely, if there are very many DB hits, we should persevere in getting more information (i.e. search constraints) from the user, depending on the penalty for longer dialogues (reflecting a 'patient vs. impatient user') and the reliability of the communication channel (the 'noise model').

4.2 Simulated Learning Environments

We test and train the strategies using RL in a simulated learning environment (see Section 3.3 for a definition). This environment includes several simulated components, such as simulated database behaviour, noise simulation, a user simulation, and particular reward functions. The individual application scenarios define how these components are instantiated. For example, if our application scenario includes an 'impatient' user, the dialogue length is punished more heavily in the reward function. For the proof-of-concept study we manually design these simulated components and scenarios to cover a wide range of (theoretically possible) applications.

4.2.1 Problem Representation

We first explain how we represent the dialogue problem we are trying to solve. Both the RL-based and hand-coded strategies rely on the same task representation. We represent the task as a 4-slot search dialogue problem. The dialogue state contains 8 binary state variables, `fill-slotN` for whether each slot number N is filled (for $1 \leq N \leq 4$), `confirm-slotN` for whether each slot number N is confirmed, and one variable `DB` for the current number of DB hits, which takes integer values between 1 and 100, resulting in $2^{10} \times 100 = 102,400$ distinct dialogue states. The following system actions are available for exploration in every state:

- `greet` e.g. "How may I help you?"
- `ask a slot` (`AskASlot`), e.g. "What kind of music would you like?"
- `explicit confirm` (`explicitConf`), e.g. "Did you say Jazz?"
- `implicit confirm and ask a slot` (`implConf-AskASlot`) e.g. "OK, Jazz music. Which artist?
- `close and present information` (`presentList`) e.g. "The following items match your query ..."

The action "`greet`" is an open-initiative question for as many search constraints as the user wishes to give: "How may help you?". The next slot name to ask (for `AskASlot` and `implConf-AskASlot`) is controlled by a process model for the domain which describes a default ordering on slots for the task (e.g. ask for music first, then artist, then album, then song title, or ask city_name, food_type, price_range, then location for restaurants). The user is not constrained to follow this ordering, and can also over-provide information using mixed-initiative behaviour such as over-answering.

4.2.2 Database Retrieval Simulations

In contrast to previous work, we do not use one specific back-end database, but we simulate two extreme cases of database behaviour in order to explore the space of possible policies. In particular, we create a "monotonic" and a "random" database retrieval model, both reflecting different ways a user query can affect search results. To the authors' knowledge this work is the first work which uses simulated databases to investigate the effects that the nature of the database, together with noise, has on policy learning.

In both cases, we assume a total database size of 100 items. We assume that presenting more than 100 results to a user is never going to be desirable, so in this study we learn strategies where there are 100 or fewer possible answers. When the number of results is more than 100, we assume that the correct strategy is to ask for more constraints from the user. We also assume that the user's goal item is contained in the database. Learning when to ask for constraint relaxation is not part of this proof-of-concept study.

4.2.2.1 Monotonic Database Simulation

The first "monotonic" database retrieval simulation models extremely well-behaved search tasks where each additional search constraint strictly reduces the number of search results obtained. Thus, if the user fills a search slot (e.g. "I want a song by Arcade Fire") the number of results returned is strictly less than in the prior state. This models boolean AND search. Conversely, if a slot becomes unfilled (e.g. by a rejected confirmation move), the number of search results will increase. The simulation draws samples from a normally distributed database. Every search constraint (i.e. filled slot) lowers the mean (μ_{db}, Equation 4.2) and narrows down the standard deviation (δ_{db}, Equation 4.2), i.e. for fewer search constraints the distribution will be flatter (number of hits clusters more widely around the mean and the curve is closer to a random distribution); the more the search becomes constrained the closer the number of hits clusters to the mean.

$$\mu_{db} = \frac{databaseSize}{2^{no.\ filledslots}} \tag{4.1}$$

$$\delta_{db} = \frac{databaseSize}{e^{no.\ filledslots}} \tag{4.2}$$

4.2.2.2 Random Database Simulation

For the second "random" database model a user-provided search constraint (slot) can either be interpreted as an AND or an OR constraint. That is, the number of DB hits can either increase or decrease. This is approximated in a random model, sampling between 1 and 100 hits. For this model newly provided information may in fact open up new possibilities in the data. For example the user might say "how about restaurants in the Old Town?" thus shifting focus away from the current set of results and opening up a new (possibly larger) set of search results. This is not intended to be a realistic retrieval model - but one which occupies one end of a spectrum. Our objective is to show that our policy learning framework covers this spectrum.

4.2.3 Noise Model

The noise model simulates the effect of channel noise on the interaction. In this book we describe noise in terms of its effect on the (estimated) task-success from the system's point of view, and its effect on the user's behaviour, i.e. the user correcting or rejecting the system's hypothesis. Thus, the noise model is implemented within two different components of the simulated environment. First, it determines how task-success is calculated in the reward function, as described in Section 4.2.5. It also affects the probabilities for generated user behaviour in the user simulation, as described in Section 4.2.4.

In particular the employed noise model reflects the chance of a filled slot (which is not yet confirmed) being correct (P_f) under the specific noise conditions. We design two different noise scenarios for the proof-of-concept study:

- High Noise (*HN*): 50% change of filled slots being correct.
- Low Noise (*LN*): 80% chance of filled slots being correct.

We further discuss this type of noise simulation in in Section 7.7. In the next sections we explain how this noise model is realised within the user simulation and the reward function.

4.2.4 User Simulations

The user simulations that we employ for the proof-of-concept study are simple bi-gram models which interact on the intention level. The possible user acts are stochastic estimates conditioned on the previous system action to simulate a relatively collaborative user behaviour. The bi-gram probabilities are manually set, as shown in Table 4.1. Note that these models solely serve the purpose of simulating dialogue behaviour for the proof-of-concept study. For a real application these frequencies are learned from data, as described in Section 7.8. Possible user actions are the following:

- `yes-answer`
- `no-answer`
- `provide-asked`, e.g. "I want ABBA" when asked "Ok, Pop music. What band do you want?"
- `provide-other`, e.g. "I want ABBA" when asked "What type of music do you want?"
- `re-provide-asked`, e.g. "no, I want Pop" when asked "Did you say Rock music?
- `provide-two` slots, e.g. "I want an ABBA song from the album Waterloo"
- remain silent

For example, if the system's previous move was to ask for a slot, the collaborative user has a 70% chance of providing the requested slot value, a 20% chance of providing a different slot value, a 6% chance of providing two slot values, and a 4% chance of remaining silent (see Table 4.1).

For system confirmation moves the likelihood of the user's reply is determined by the noise model (i.e. the probability of a filled slot being incorrect (P_f)). In particular, the noise model determines the chance of the user accepting or rejecting the system's hypothesis. After a confirmation the following actions are interpreted as an acceptance: `yes`, `provide-asked`, `provide-other`, `provide-two`. The user acts `no` and `re-provide-asked` are interpreted as a rejection. For example, when the system asks for an explicit confirmation under high noise conditions ($Pf = 50\%$), the user simulation accepts the system's hypothesis with 48% probability in total. The user model accepts an hypothesis by either performing a `yes` act or providing another slot value or another two slot values. Thus, the total probability for accepting is split over these acts. In 48% of cases the user rejects the system's hypothesis. The user model rejects an hypothesis by either performing a `no` act or re-providing the slot value). 2% of each likelihood for acceptance and rejection are assigned to the user being silent (i.e. 4% in total).

sysAct/ userAct	yes	provide-other	provide-asked	re-prv-asked	no	provide-two	silent
greet	–	80.0	–	–	–	16.0	4.0
askASlot	–	20.0	70.0	–	–	6.0	4.0
explConf (*HN*)	40.0	4.0	–	8.0	40.0	4.0	4.0
explConf (*LN*)	50.0	4.0	–	8.0	10.0	4.0	4.0
implConf (*HN*)	2.0	2.0	40.0	30.0	18.0	4.0	4.0
implConf (*LN*)	4.0	4.0	60.0	3.0	15.0	10.0	4.0

Table 4.1 Probabilistic user simulation with noise model High Noise (*HN*) and Low Noise (*LN*)

Note that a similar study was conducted by Lemon and Liu (2007). Here the authors show that the RL framework also spans non-collaborative user behaviour. In the current study, however, we focus on collaborative user models, as these are more similar to the user behaviour we learn from data (see Section 7.8).

4.2.5 Objective and Reward Function

We propose a novel reward function which incorporates noise modelling. In related work the reward function is also called the *objective function* (Paek, 2006; Walker, 2005) to express the fact that it serves two different purposes: it is used as a *reward function* to train RL-based policies, as well as being used as an *evaluation function* to score the generated dialogues. We also evaluate and tune the hand-coded strategies using this function. For the proof-of-concept study we select the factors of the reward/objective function by hand. For our ultimate application we will learn these factors (as well as their relative weights) from data using the PARADISE framework (see Section 7.9). For the proof-of-concept study we define for each dialogue:

$$FinalReward = completionValue - dialogueLengthPenalty \qquad (4.3)$$
$$-DBhitsPenalty; \qquad (4.4)$$

Where $dialogueLengthPenalty$ penalises every system turn (via a TurnPenalty TP per turn) and $DBhitsPenalty$ penalises every item which is presented to the user (via an itemPenalty IP per presented item).

The $completionValue$ implements the noise model for the objective function. It is defined as the percentage probability that the user goal is in the presented result set. For example, if we know with 100% certainty that the user wants Sushi in the Old Town (i.e. 2 slots, both confirmed at 100% probability), then we have a 100% chance of supplying the user with an item that meets their goal. On the other hand, if we are in the same situation but we are only 80% sure that they want Sushi (due to channel noise) then the probability is only $.8 \times 1 \times 100 = 80\%$. Thus the completion value of a dialogue is directly related to the probability of search slots being correctly filled, which is in turn related to the noise conditions (see Section 4.2.3) under which the dialogue is being conducted. Thus, where P_c is the probability of a confirmed slot being correct, and P_f is the probability of a filled slot being correct, where C and F are the number of confirmed slots and filled (but not confirmed) slots respectively, we have:

$$completionValue = 100 \times (P_c)^C \times (P_f)^F \tag{4.5}$$

For example, in a High Noise environment, we might set $Pc = 1.0$ and $Pf = 0.5$, reflecting the fact that in a noisy environment unconfirmed slots are fairly likely to be incorrect (50% chance). The maximum possible reward for any dialogue is 100 (where $TP = 0$, $IP = 0$ and all slots are confirmed in cases where $Pc = 1$). For an example of the computation of $FinalReward$, consider a 4-slot problem where, turnPenalty (TP) =1, itemPenalty (IP) =10, and where 2 items have been presented to the user at the end of the dialogue after 6 system turns. In the case where 4 slots are filled, but only 3 were confirmed, we would then have (in the same High Noise model as above):

- $completionValue = 100 \times 1^3 \times 0.5^1 = 50$
- $dialogueLengthPenalty = 6 * 1 = 6$
- $DBhitsPenalty = 2 \times 10 = 20$
- and so ultimately, $FinalReward = 50 - 6 - 20 = 24$

4.2.6 Application Scenarios

In the proof-of-concept study we develop policies for a wide range of possible application scenarios. In order to define an 'application scenario' we choose 4 factors with 2 options each ($2^4 = 16$ scenarios in total).

DB retrieval: The database retrieval mechanism influences whether a newly-provided constraint narrows or widens the space of retrieved candidates. In Section 4.2.2 we defined the *monotonic* and the *random* retrieval models.

Noise: The level of noise influences task success and the need for confirmation strategies. The level of noise reflects the chance of a filled slot being correct (P_f), as defined in Section 4.2.3. We defined the *High Noise* and the *Low Noise* environments above.

User type: The user type influences how dialogue length is evaluated ($dialogue-LengthPenalty$). The user type is different from the user simulation, as it defines properties for this specific user, whereas the user simulation is meant to be a general model of user behaviour (see Section 7.8.1 for a definition). We define two different user types. For a *patient* user we assume that s/he does not mind very much if the dialogue gets longer (low turn penalty = -1). For an *impatient* user we assume that short dialogues are most important (high turn penalty = -10).

Screen size: The screen size influences how the length of the presented list is evaluated ($DBhitsPenalty$). For an *in-car* environment we assume that the screen size is small and the user's eyes are busy; thus lists should be as short as possible (high hit penalty = -10). [1] For the *in-home* environment we assume that the screen size is bigger and the user can spend all their attention looking at the screen; thus lists can also be longer (low hit penalty = -1).

The application scenarios given above are now reflected in how we formulate the evaluation functions for policy learning and policy testing. In particular we use the following 8 objective functions for both monotonic and random DB retrieval models (resulting in 16 scenarios):

1. *HiNoise,HiHit,HiTurn*: 50% chance of filled slots being correct, turn penalty=-10, hit penalty=-10; e.g. noisy channel, in-car/small screen, impatient user
2. *HiNoise,LowHit,HiTurn*: 50% chance of filled slots being correct, turn penalty=-10, hit penalty=-1; e.g. noisy channel, in-home/large screen, impatient user
3. *HiNoise,HiHit,LowTurn*: 50% chance of filled slots being correct, turn penalty=-1, hit penalty=-10; e.g. noisy channel, in-car/small screen, patient user
4. *HiNoise,LowHit,LowTurn*: 50% chance of filled slots being correct, turn penalty=-1, hit penalty=-1; e.g. noisy channel, in-home/large screen, patient user
5. *LowNoise,HiHit,HiTurn*: 80% chance of filled slots being correct, turn penalty=-10, hit penalty=-10; e.g. reliable channel, in-car/small screen, impatient user
6. *LowNoise,HiHit,LowTurn*: 80% chance of filled slots being correct, turn penalty=-1, hit penalty=-10; e.g. reliable channel, in-car/small screen, patient user
7. *LowNoise,LowHit,HiTurn*: 80% chance of filled slots being correct, turn penalty=-10, hit penalty=-1; e.g. reliable channel, in-home/large screen, impatient user
8. *LowNoise,LowHit,LowTurn*: 80% chance of filled slots being correct, turn penalty=-1, hit penalty=-1; e.g. reliable channel, in-home/large screen, patient user

The dialogue design task is now to find the right “trade-offs” for list size and dialogue length, as well as deciding when to ask and confirm slot values in order to optimise the returned value from the reward/objective function. We apply two different techniques to define dialogue strategies according to the given objective functions: First, we manually tune thresholds for a rule-based strategy to maximise the return from the objective function. Second, we directly employ the objective functions to train RL-based dialogue policies.

4.3 Threshold-based Baseline

First, we construct a hand-coded baseline strategy for each of the dialogue scenarios. Our baselines are “state of the art” policies in the sense that they allow mixed-initiative interaction and use thresholds (for dialogue length and number of DB hits) chosen for the particular operating environments, similar to the system described in Varges et al (2006). We manually tune different thresholds to maximise the returns from reward functions as listed in the last section. We follow a similar method to the one suggested by Chung (2004a): we manually set several combinations of different thresholds and test their performance with the simulated user. We repeat this until

[1] We use linear functions for the list size penalties, i.e. assuming that the shorter lists are the better. In Section 7.9 non-linear functions will be explored.

we reach the maximum performance. This "hand-tuning" partially addresses one of the criticisms made by Paek (2006), that learned polices are usually not compared to hand-coded policies that have been designed for the same conditions.

The confirmation/grounding mechanism (reflecting the need to confirm in a high-/low noise environment) is not controlled by thresholds (since we do not simulate ASR confidence score, see discussion Section 7.7), but is controlled by the following general algorithm (which is then modified by the scenario-specific thresholds):

1. `Greet` the user,
2. either `AskASlot` (if no slots need to be confirmed) or `ImplConf-AskASlot` (if there are remaining slots to ask and slots to confirm),
3. then repeat 2 until there are no slots left to fill,
4. then `ExplConf` (explicitly confirm) the remaining filled slots (if any)
5. `PresentInfo`; present the answer set to the user.

The hand-coded baselines will thus always `PresentInfo` if all slots are confirmed or the number of items returned from the database is less than a threshold (for example < 7) or the dialogue length exceeds a threshold (for example > 6). The different thresholds for the hand-coded policies thus determine when to stop asking for new constraints. For the baseline systems, we designed four hand-coded policies with the following combinations of hand-tuned thresholds:

H_{ss}: short dialogues (≤ 6), short lists (≤ 7): e.g. in-car, with impatient user.
H_{ls}: long dialogues (≤ 10), short lists (≤ 7): e.g. in-home, with patient user.
H_{sl}: short dialogues (≤ 6), long lists (≤ 14): e.g. in-car, with impatient user.
H_{ll}: long dialogues (≤ 10), long lists (≤ 14): e.g. in-home, with patient user.

Example 4.2 shows an example dialogue using the simulated user and a baseline policy with DB information presentation and dialogue length thresholds 7 and 6 respectively. At each system move we report the number ("db:") of DB hits obtained by the system. [2]

Example 4.2.

```
System: ``Hello, how may I help you?'"
          greet
User:  ``Do you have some modern classical music?'
          provide_info(slot1)
System: ``Ok, modern classical. Which composer?''
          implConf(slot1)AskASlot(slot2) (db: 77)
User:  ``Steve Reich?"
       provide_info(slot2)
System: ``Do you have a preferred piece by Steve Reich?"
      implConf(slot2) AskASlot(slot3) (db:25)
User: ``Yes, Drumming"
      provide_info(slot3)
System: ``Did you say Drumming?"
```

[2] Note that our database simulation assumes continuous values in the database and does not account for domain-specific constraints such as album size, as the proof-of-concept study is meant to be domain independent.

```
      explicitConfirm(slot3) (db: 4)
User:  ``Yes"
      yes_answer
System: ``OK, there are 4 parts on this album,
    which are now shown on the screen."
    presentList(slot1, slot2,slot3)(db:4)
```

A major draw-back of hand-coded thresholds for a rule-based systems is that they are usually globally defined, i.e. the thresholds are fixed for the whole interaction. Ideally one wants to formulate rules like “if the dialogue gets too long, relax the constraint for DB presentation”, or “if the threshold for DB hits is already reached at the beginning of the dialogue, it might still be worth asking the user for more confirmation/constraints in order to increase task success”, and so on. Specifying these contextual rules makes hand-coding even more complex and it becomes more difficult to design a policy which is optimal (with respect to the defined objective function).

The Reinforcement Learning experiments presented below effectively learn optimal values for different objective functions.

4.4 Reinforcement Learning Method

For policy learning we use the REALL-DUDE development tool (Lemon et al, 2006c), which implements the SARSA RL algorithm (see Section 3.2.2.2) with linear function approximation (see Section 3.2.2.3). We will describe the REALL-DUDE learning system in more detail in Section 7.11. For learning, the dialogue is formalised as a Markov-Decision-Process (MDP), with the problem formalised using the state-action space as described in Section 4.2.1.

In the following we report on 16 experiments where we systematically vary the 2 database definitions and 8 objective/reward functions (as defined in Section 4.2.6).

4.4.1 Training the Policies

For learning we use the following parameters:

- number of cycles = 96,000
- learning rate α = 0.2
- discount rate γ = 0.95
- eligibility trace parameter λ=0.9
- exploration halving time $\varepsilon = \frac{1}{6} \times numberofcycles = 16,000$

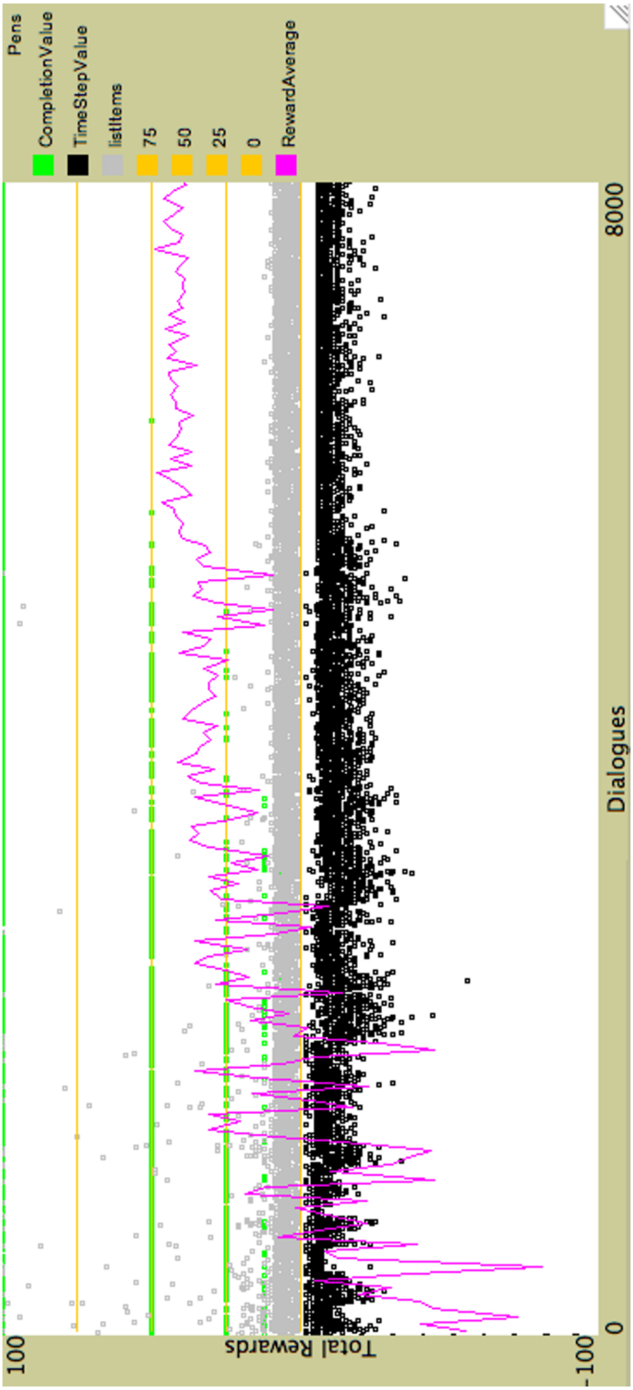

Fig. 4.2 Learning curve for training $HiNoise, HiHit, LowTurn$ policy, Monotonic DB

The learning rate α is the step-size parameter for learning, as described in Section 3.2.2.2. The discount factor gamma (γ) may take a value between 0 and 1. For the dialogue management problem we set it close to 1 so that future rewards are strongly weighted (Sutton and Barto, 1998). The lambda (λ) parameter refers to the trace decay parameter, with $0 \leq \lambda \leq 1$. Higher settings lead to longer lasting traces; that is, a larger proportion of credit from a reward can be given to more distant states and actions, (with λ=1 producing parallel learning to Monte Carlo algorithms, see Section 3.2.2.2). The exploration halving time epsilon (ε) halves the exploration rate every 16k dialogues, where ε is about $\frac{1}{6}$ of the agent's total number of cycles. Each policy is trained over 96,000 cycles of the system, which resulted in about 12,000 simulated dialogues per training run. Figure 4.2 shows a training run where the policy learns to reduce the number of database hits presented to the user (grey, +1 per presented item) and dialogue length (black, -1 per turn) while obtaining a high completionValue (green). Average dialogue reward, computed over windows of 50 dialogues, is shown by the red line. Note that after about 6000 dialogues the learner has settled on a policy of confirming all information slots while presenting fewer than 10 items in fewer than 20 turns.

4.5 Results

We test each (learned and baseline) policy in each condition by running 550 test dialogues in simulation. [3] We compare the policies in respect of their average final reward per dialogue over the test runs. We then perform a paired t-test (with Bonferroni correction) on the final rewards, to determine statistical significance.

The results produced by the learned policies (denoted RL) and the best hand-coded baseline can be seen in Table 4.2. The manually optimised hand-coded strategy performs equally well to the RL-based strategy in only 4 out of 16 conditions. Especially in the more challenging random DB case, the RL-based policy learns to adapt to the current context better. Detailed results produced by the learned policies and all the different hand-coded baselines ($H_{ss} \ldots H_{ll}$) for 4 of the operating conditions/application scenarios can be seen in Figure 4.3. We further discuss these results below.

[3] Note that this policy is trained and tested on the same user simulation. Although this method is commonly employed for dialogue strategy learning, e.g. (Heeman, 2007; Henderson et al, 2008) etc., training and testing on the same user simulation has been criticised of being "cheating" (Paek, 2006). In Chapter 7 we train and test a RL-based policy using two different user simulations. We also show that results transfer between user simulations, i.e. there is no significant difference in strategy performance between the user model which the strategy is trained on and the one which is was only used for testing (see Section 7.11.2).

Reward def.	best hand-coded	RL
	Monotonic DB	
$HiNoise, HiHit, HiTurn$	-31.1±54.7	28.1±7.8***
$HiNoise, LowHit, HiTurn$	25.3±30.2	28.4±11.1**
$HiNoise, HiHit, LowTurn$	34.9±30.6	41.3±21.3***
$HiNoise, LowHit, LowTurn$	85.7±11.2	86.0±4.6
$LowNoise, HiHit, HiTurn$	85.6±5.9	87.2±4.5**
$LowNoise, HiHit, LowTurn$	38.1±28.7	41.8±45.9
$LowNoise, LowHit, HiTurn$	31.8±14.4	31.3±10.1
$LowNoise, LowHit, LowTurn$	-22.6±54.3	-20.2±22.5
	Random DB	
$HiNoise, HiHit, HiTurn$	-373.3±321.4	-159.7±96.3***
$HiNoise, LowHit, HiTurn$	-275.1±312.0	-115.7±164.9***
$HiNoise, HiHit, LowTurn$	-1.3±38.8	13.8±25.5***
$HiNoise, LowHit, LowTurn$	55.2±30.8	62.6±21.6***
$LowNoise, HiHit, HiTurn$	55.1±32.3	80.5±9.3***
$LowNoise, HiHit, LowTurn$	-290.2±310.7	-155.3±217.5***
$LowNoise, LowHit, HiTurn$	1.9±39.5	20.4±23.2***
$LowNoise, LowHit, LowTurn$	-333.6±316.5	-166.1±206.9***

Table 4.2 Direct comparison between mean rewards (with standard deviation ±) obtained by the RL-based and best performing hand-coded policy *for each environment* with results from a paired t-test; ** denotes $p < .005$, *** $p < .001$

Consider, for example, the pair of graphs in the top row of table 4.3. These show the differences between the monotonic and random DB cases, for the case $HiNoise, LowHit, HiTurn$. In the monotonic case, we can see that the learned policy has the fewest turns, but presents the longest lists, and outperforms all the hand-coded configurations. All policies perform worse in the more challenging random DB case, as expected, but the learned policy for the random case has learned that in some cases not all filled slots need to be confirmed (presumably because sometimes users provide new information when asked for confirmation, which can increase the number of DB hits in the random model), and to keep dialogues short. The learned policy here significantly outperforms all the hand-coded configurations (at $p < .001$ or $p < .01$).

Now consider the first column of table 4.3. These two graphs show the differences between the $LowHit, HiTurn$ and $HiHit, LowTurn$ cases for monotonic DBs in $HighNoise$. Moving from the $LowHit, HiTurn$ (top left) to the $HiHit, LowTurn$ (bottom left) case, we see that the learner earns more reward by increasing dialogue length and decreasing the number of presented items, as expected. In both cases the learned policies significantly outperform all the different hand-coded (threshold-based) configurations ($p < .001$).

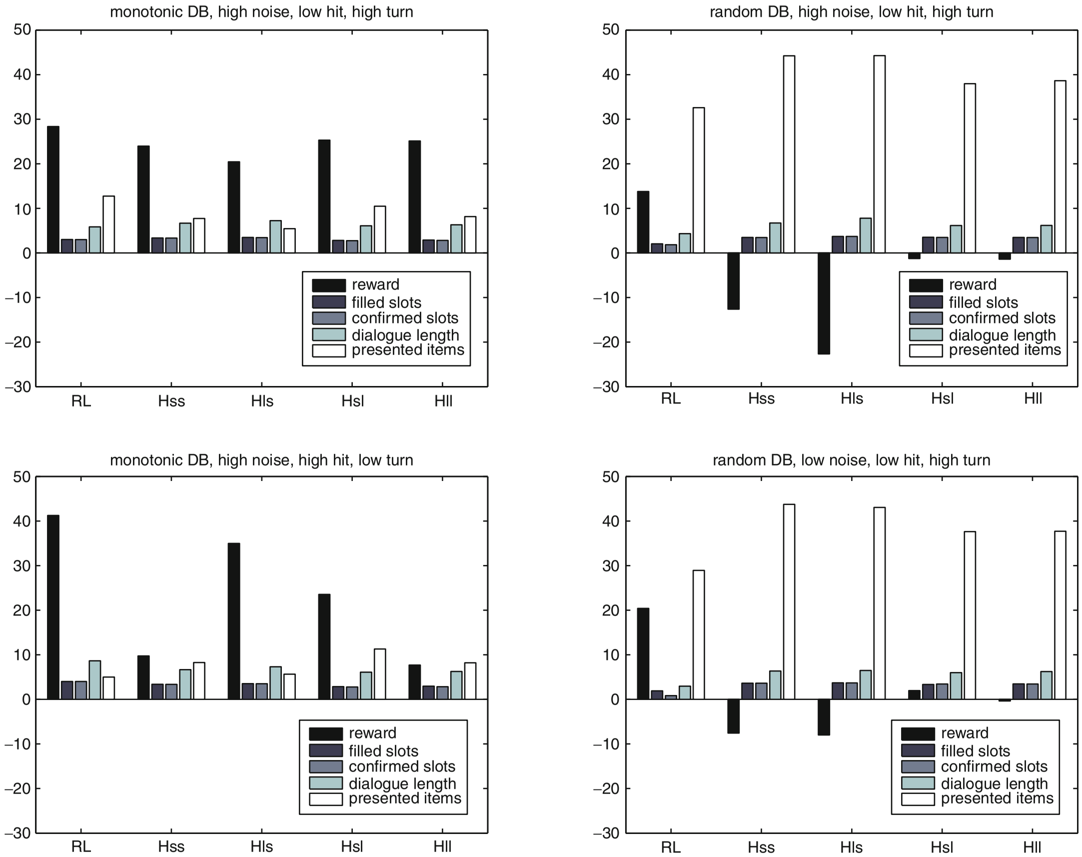

Table 4.3 Comparison of mean performance measures for learned and hand-coded policies, for monotonic (left) and random (right) DB models

Finally, consider the top right ($HiNoise, LowHit, HiTurn$) and bottom right ($LowNoise, LowHit, HiTurn$) graphs, which are both for the random DB model. Moving down, we see that in the LowNoise case, the learner has settled on a policy of confirming fewer slots than in the high noise case (because filled slots are more reliable in low noise). Interactions are thus shorter, and more reward is gained. In both cases the learned policies significantly outperform all the hand-coded configurations ($p < .001$).

The learned policies produce an average *relative* increase in reward of 86.78% over the various hand-coded baseline policies. In 93% of the cases the learned policies perform highly significantly better than the hand-coded ones ($p < .001$).

It is also interesting to note that amongst all the factors describing our application scenarios the DB model was the one with with the strongest impact on the strategies' performance. A sensitivity analysis using stepwise linear regression analysis on performance measures using noise levels, user type, screen size, and DB retrieval type as predictors showed that *DB retrieval* is the most important factor, see Table 4.4.

factors	B	Std. Error	β
constant	6.44	.101	
DB retrieval	-1.8	.034	-.57***
noise level	-0.25	.013	-.239***

Table 4.4 Multiple regression analysis on factors influencing strategy performance (reward); $R^2 = .4$, dismissed factors: user type, screen size

We now provide a qualitative description of the learned policies, using examples from the simulated corpus. We report the number of logged database hits at each stage of the interaction ("db:"). In example 4.3 the policy trained for monotonic DB and $HiNoise, LowHit, HiTurn$ has learned to attempt to fill all 4 slots and confirm them before presenting items. It has also learned not to explicitly confirm if the user rejects, if the dialogue gets too long. This strategy prefers to present a list of results not including the rejected slot as an search constraint. For the $HiNoise$ environment presenting a list of options rather than asking for explicit confirmation circumvents the speech recognition problem.

Example 4.3.

```
System: ``Hello, how may I help you?''
         greet
User:   ``I want a Radiohead song''
         provide_info(slot1)
System: ``Ok, a Radiohead song. From what album?''
         implConf(slot1)AskASlot(slot2) (db: 77)
User:   ``From OK Computer''
         provide_info(slot2)
System: ``OK from OK Computer. What song title?''
         implConf(slot2)AskASlot(slot3) (db: 12)
User:   ``Paranoid Android''
         provide_info(slot3)
System: ``Did you say Karma Police?''
         explicitConfirm(slot3) (db: 1)
User:   ``No''
         no_answer
System: ``OK, there are 12 items you might be looking for,
         which are now shown on the screen.''
         presentList(slot1,slot2)(db:12)
```

Example 4.4 shows an example of the policy learned for random DB, *LowNoise*, *LowHit*, and *HiTurn*. The learned policy here fills one slot and then immediately presents the list, since it has already found a relatively low number of hits, and for a random DB filling more slots does not necessarily result in a lower number of DB hits. Due to the *LowNoise* condition it has also learned not to confirm the slot in this situation. This strategy has learned to "take its chances" in this type of situation, once a relatively small number of results has been retrieved.

Example 4.4.

```
System: ``Hello, how may I help you?''
        greet
User:    ``I want a Radiohead song''
        provide_info(artist_name)
System:   ``OK, there are 22 items you might be looking for,
             which are now shown on the screen.''
        presentList(artist_name,album_name)(db:22)
```

4.6 Summary

In this Chapter we provide the general proof-of-concept that RL-based strategies significantly outperform manually tuned heuristics for a wide range of possible application scenarios. The RL-based policies produce an average *relative* increase in reward of 86.78% over the various hand-coded baseline policies. In 93% of the cases the learned policies perform highly significantly better than the hand-coded ones ($p < .001$).

We also demonstrated how to use a statistical policy learning framework (RL with the SARSA algorithm and linear function approximation) to address a complex problem: what questions to ask the user, how many database search results to present to the user, and when to present them, given the competing trade-offs between the length of the answer list, the length of the interaction, the type of database, and the noise in the speech recognition environment.

One of the major strengths of RL-based strategies is that they "intelligently" adapt their behaviour to the changing dialogue context in order to satisfy an overall long-term objective. One can think of this as locally adaptive "thresholds" which are globally optimised. The standard technique for hand-coded threshold-based policies, in contrast, is to set a single "global" threshold which does not change over the whole interaction.

This Chapter also demonstrates that RL-based strategies can learn to adapt to the simulated learning environment in very subtle ways. Thus, the correct representation of the learning environment is one of the key challenges for this framework. When constructing a simulated learning environment by hand, one might argue that the problem has only been transferred to another level of manual tuning. Data-driven methods to construct such a learning environment should therefore be preferred. In cases where a system is designed from scratch, however, it is often the case that no suitable in-domain data is available. Collecting dialogue data without a working (hand-crafted) prototype is problematic, leaving the developer with a classic chicken-and-egg problem. In the next chapter we introduce an approach where the entire simulated learning environment is "bootstrapped" from Wizard-of-Oz data.

Part II
Policy Learning in Simulated Environments

Chapter 5
The Bootstrapping Approach to Developing Reinforcement Learning-based Strategies

Fig. 5.1 Baron Münchhausen escaping from a swamp by pulling himself up by his own hair; after a novel by Raspe (1785), illustration by Hosemann (1840) – The first 'bootstrapping'

This Chapter motivates and introduces a procedural method for automatic strategy learning from Wizard-of-Oz (WOZ) data. Note that this Chapter presents the

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_5,

approach on a conceptual level, while the concrete practical realisation of the individual steps will be further elaborated in the subsequent chapters.

It should also be noted that the presented steps are not unique to the method introduced in this book, but most of them are required for any simulation-driven approach to strategy learning (though the last step is specific to our method). A contribution of this book is that all these steps are now performed starting with a limited WOZ data set, and that specific methods are introduced to build and validate the obtained simulations.

5.1 Motivation

Statistical learning approaches, such as Reinforcement Learning (RL), for Spoken Dialogue Systems offer several potential advantages over the standard rule-based hand-coding approach to dialogue systems development (as further explained in Chapter 3.1): a data-driven development cycle, provably optimal action policies, a precise mathematical model for action selection, possibilities for generalisation to unseen states, and automatic optimisation of competing trade-offs in the objective function.

In the previous Chapter we showed that RL-based strategies outperform hand-coded strategies with manually tuned thresholds for a wide range of application scenarios. One of the major strengths of RL-based strategies is that they can "intelligently" adapt their strategies to the (local) representation of the dialogue environment in order to satisfy an overall objective. Thus, the correct representation of the learning environment is one of the key challenges for this framework, and data-driven methods to construct such an environment should be preferred over hand-crafting (as done for the proof-of-concept study in Chapter 4).

One of the major limitations of this approach is that it relies on a large quantity of data being available. In cases when a fixed data set is used for learning, e.g. (Henderson et al, 2008; Singh et al, 2002; Walker, 2000), the optimal policy can only be discovered when it is present in the data set. [1] To overcome this problem, simulated learning environments are being used to explore optimal policies which were previously unseen in the data, e.g. (Ai et al, 2007b; Eckert et al, 1997; Young et al, 2009). However, several aspects of the components of these simulated environments are usually hand-crafted, and thus limit the scope of policy learning. In particular, the optimization (or reward) function is often manually set (Paek, 2006). In order to build simulation components from real data, annotated in-domain dialogue corpora have to be available, which explore a range of dialogue management decisions. Collecting dialogue data without a working prototype is problematic, leaving the developer with a classic "chicken-and-egg" problem. We therefore propose to learn

---

[1] Note, by a policy being "present in a data set" we mean that the set of state-action mappings which define the policy is contained in that data set. When a policy is not present in a data set, either some states covered by the policy are not seen at all in that data, or the actions chosen by the policy in some states are different to those seen in the data.

dialogue strategies using simulation-based RL, where the simulated environment is learned from small amounts of Wizard-of-Oz (WOZ) data.

In contrast to preceding work, our approach enables strategy learning in domains where no prior system is available. Optimised learned strategies are then available from the first moment of online-operation, and handcrafting of dialogue strategies is avoided. This independence from large amounts of in-domain dialogue data allows researchers to apply RL to new application areas beyond the scope of existing dialogue systems. We call this method "bootstrapping".

In addition, our work is the first using a data-driven simulated environment. Previous approaches to simulation-based dialogue strategy learning usually handcraft some of their components.

Of course, some human effort is needed in developing the WOZ environment and annotating the collected data, although automatic dialogue annotation could be applied (Georgila et al, 2009). The alternative – collecting data using handcoded dialogue strategies – would still require annotation of the user actions, and has the disadvantage of constraining the system policies explored in the collected data. Therefore, WOZ data allows exploration of a range of possible strategies, as intuitively generated by the wizards, in contrast to using an initial system which can only explore a pre-defined range of options.

5.1.1 Term Definition

The term to 'bootstrap' has various different meanings in the fields of computer science (to 'boot' a system), linguistics (theory of language acquisition), statistics (sample with replacement for statistical inference), physics, law, business and many more. We use the term in a sense which is closer to its original meaning of "pulling oneself up by one's own bootstraps". The term is said to have come from a tale from the adventures of Baron Münchhausen who, according to the story, escaped from a swamp by pulling himself up by the straps of his boots. Although in other versions of the story he hoisted himself using his own hair (Raspe, 1785), see Figure 5. In this book the term is used to describe the problem of how to learn an optimal strategy before a working system or prototype exists, and thus circumvent the chicken-and-egg problem. The bootstrapping method conceptually contrasts with dialogue strategies being manually "uplifted" by a by a human expert.

Note that "bootstrapping" was also used to determine other aspects of dialogue system design: Weilhammer et al (2006) "bootstrap" the ASR language model from WOZ data. Fabbrizio et al (2004, 2008) "bootstrap" various dialogue components from out-of-domain data. In general, the term "bootstrapping" is used to describe the problem of how to build system components in a data-driven manner without having in-domain data available.

5.1.2 Related Work

In general, RL-based dialogue policy learning is based on trial-and-error learning from seeing as many interactions as possible (see Sections 2.3.4 and 3.2.2.3). There are three main techniques for addressing this problem: generalisation of state spaces, e.g. (Henderson et al, 2005, 2008) , rapid learning from small data sets (Gasic and Young, 2011; Pietquin et al, 2011b), and simulation-based learning, also see Section 2.3.4. Nevertheless, both techniques require initial data to start with. In cases where a system is designed from scratch, however, there is often no suitable in-domain data.

Learning dialogue strategies from human-human interaction data (if available) is not an option, since humans behave differently with machines than with other people (Doran et al, 2001; Jönsson and Dahlbäck, 1988; Moore and Morris, 1992). Furthermore, human-human interaction is usually less affected by channel noise since humans are much better at handling noise and uncertainty (see Section 2.1).

It is also not considered an option to use several disparate data sets from which to build different simulated components. This approach assumes that simulated dialogue components are independent from each other, which clearly is not the case for most of the simulations. For example, it assumes that the user behaviour is independent from the channel noise. It is common practise in current research to obtain all the simulated components from one data set, e.g. (Henderson et al, 2005, 2008; Schatzmann, 2008; Singh et al, 2002). The work of (Prommer et al, 2006) also experiments with using data from an isolated data collection to retrain some of the simulated components. However, the results indicate that this can easily lead to experimental conditions which are inconsistent with the environment of the final task setup. Nevertheless, some sub-components can be learned using out-of-domain data. For example, a phoneme confusion matrix for ASR modeling can be learned on any (big and representative enough) data set (Stuttle et al, 2004).

In sum, there is a strong indication that we need one consistent data set of human-machine interaction for building a simulated learning environment. When building a system from scratch, however, there is often no suitable in-domain data available. Different approaches have been suggested to address the problem of lacking initial training data: handcrafting the simulated components (Schatzmann et al, 2007a), online learning (Chickering and Paek, 2007), transfering policies from another domain (Lemon and Liu, 2007; Lemon et al, 2006a), and also starting learning from a limited amount of WOZ data (Prommer et al, 2006; Williams and Young, 2004b).

Schatzmann et al (2007a) suggest to manually set the initial parameters of a simulated environment to learn a policy. This policy is then used to gather initial data, which then can be used to re-train the parameters of the simulation and then re-train the policy. We argue that that data-driven methods should be preferred in order to ensure consistent, realistic behaviour (see discussion Section 4.6).

Chickering and Paek (2007) circumvent the data problem by using online RL. However, online learning requires many iterations with real users to have noticeable effect. In addition, it requires the continuous use of exploration during online operation, which can result in a quite confusing and frustrating interaction for the system's users. Furthermore, it is not clear how to infer the reward function during online operation. Hence, online learning can be used to constantly improve/adapt a reasonably well-performing strategy, but it is not suited to design a strategy from scratch.

In recent work the use of WOZ data has been proposed in the context of Reinforcement Learning (Levin and Passonneau, 2006; Prommer et al, 2006; Williams and Young, 2004b). Williams and Young (2004b) use WOZ data to discover the state and action space for the design of a Markov Decision Process (MDP). Prommer et al (2006) use WOZ data to build a simulated user and noise model for simulation-based RL. While both studies show promising first results, their simulated environments still contain many hand-crafted aspects, which makes it hard to evaluate whether the success of the learned strategy indeed originates from the WOZ data. Schatzmann et al (2007a) propose to 'bootstrap' with a simulated user which is entirely hand-crafted. In the following we propose what is currently the most strongly data-driven approach to these problems. We also show that the resulting policy performs well for real users.

In the following sections we discuss the general advantages and challenges of RL-based dialogue strategy learning when the simulated learning environment is obtained from WOZ data.

5.2 Advantages for Learning from WOZ Data

There are several advantages when learning RL-based dialogue strategies from WOZ data, which makes this approach an attractive alternative to the other approaches outlined above.

First of all, the data collection in a WOZ experiment does not require a working prototype, as discussed before. This allows us to learn optimal strategies for domains where no working dialogue system already exists. Optimised learned strategies are then available from the first moment of online-operation, and handcrafting of dialogue strategies is avoided. This independence from large amounts of in-domain dialogue data allows researchers to apply RL to new application areas beyond the scope of existing dialogue systems.

Furthermore, WOZ data can be used to explore human strategies as a basis for automatic optimisation. For example, the state and action set for learning can then be defined based on observing human behaviour first, as also suggested by Williams and Young (2004b) and Levin and Passonneau (2006). A hand-coded strategy for data-collection would not explore different dialogue policies. Even if some random elements could be added into a hand-coded strategy, this would still be constrained variation. Use of human wizards allows less restricted exploration of dialogue policies.

In addition, a WOZ experiment includes a controlled experimental setup which can be designed to closely anticipate the final system setup. Subjects are also asked to fill out a questionnaire (see Section 3.3.1). In conventional system development these user ratings are analysed by an expert and used to guide the design process. Alternatively, these scores can also be utilised to learn an automatic evaluation function using the PARADISE framework (see Section 2.2.2). In RL-based strategy development this function can then be directly used to train and test the dialogue strategy.

5.2.1 Challenges for Learning from WOZ Data

However, using WOZ data for RL-based strategy development also generates some challenges. In the following we discuss two problems in particular: how to learn simulated components from small data sets, and how to account for the fact that a WOZ study only simulates real Human-Computer Interaction.

First of all, WOZ experiments are expensive and usually only produce small amounts of data. Thus, the bootstrapping approach requires methods to learn simulated environments from little data. In order to find methods which are "appropriate" one needs to remind oneself of the purpose of a simulated environment for strategy learning. This purpose is usually two-fold. On the one hand it has to provide *realistic* feedback to the learner. This "experience" gathered in the simulated environment is then reflected in the learned Q-function (see Section 3.2.2). On the other hand, a simulated environment also needs to *cover* a wide state-action space in order for the learner to explore all possible situations in the dialogue. Exploration is necessary to guarantee robust strategies (see Section 2.2.4).

In previous work, the simulated components of these learning environments are either hand-crafted or learned via Supervised Learning (SL). The problem when applying SL to small data set is that the resulting models often suffer from "data sparsity", which is defined as follows Alpaydin (2004).[2] In general, SL is learning to infer a general mappings from seen example (see Section 2.3.1). If training examples are rare, the learner tends to "overfit" the data, i.e. it adjusts to very specific features of the training data. Models which overfit do not generalise well to unseen events and are also not realistic, in a sense that they are only representing the behaviour of a small population. Thus, the major challenge when constructing simulated learning environments from small amounts of WOZ data is to construct simulated components which generate realistic and wide-coverage behaviour.

Previous work on learning simulated environments from WOZ data used handcrafted components in combination with "low conditioned" models (Prommer et al, 2006), i.e. models which do not cover the entire policy space. For example, the bigram user simulation used by Prommer et al includes a lot of zero frequencies (see Section 7.8.1.2). In Chapter 7 we will introduce appropriate methods for building and evaluating simulated components from small data sets.

Another major problem in this framework is that a WOZ study only simulates real HCI. That is, a simulated environment learned from this data is a "simulation of

[2] Note that corpus size is not the only factor which can cause data sparsity. A corpus also need to be representative of the real underlying distribution, which does not necessarily depend on the size.

a simulation". We thus explicitly show that the learned simulations are a sufficient estimate of real HCI by post-evaluating the simulated environment in Section 8.5. We also show that a policy trained in this simulated environment "transfers" to real HCI, i.e. we test whether the obtained results are compatible. We do this by including an extra post-evaluation step in the general simulation-based RL framework.

In sum, developing RL-based dialogue strategies from WOZ data offers a number of important advantages over previous approaches, such as learning without a working prototype, the ability to study wizard behaviour, and the availability of user ratings. The major challenges are the data sparsity problem for SL and the fact that a WOZ study only mimics real HCI. In the next Section we describe a general method which addresses these challenges.

5.3 The Bootstrapping Method

In this book we introduce a method for training an optimised policy from WOZ data. We call this method bootstrapping, because an optimised strategy exists even before a working prototype system (see Section 5.1.1).

In particular, we follow a 5-step procedure (see Figure 5.3): We start by collecting data in a WOZ experiment. From this data we train and test different components of our simulated environment (such as the noise simulation and the simulated user) using Supervised Learning techniques. We then train and evaluate dialogue policies by interacting with the simulated environment using Reinforcement Learning. Once the learned policies are "good enough" we test them with real users. In addition, we introduce a final phase where we explicitly evaluate whether the models and policies obtained by bootstrapping from WOZ data are a valid estimate of real HCI (see step 5 in Figure 5.3).

The next Sections provide an overview of the specific methods used in each of the steps. We also include pointers to the respective Chapters which report on these steps in more detail.

5.3.1 Step 1: Data Collection in a Wizard-of-Oz Experiment

The first step in our framework is to collect data in a WOZ experiment. In order to use WOZ data as an initial corpus for dialogue strategy learning we apply the following changes to the conventional WOZ setup (as described in Section 3.3.1).

First of all, we are not only interested in the users' behaviour, but also what kind of strategies human wizards apply. That is, the wizard also becomes a subject of our study. (This approach has also been described as a "Ghost in the Machine" method.) We therefore have several different wizards participating in our study and explore "intuitive" strategies applied by the wizards. We do not restrict the wizard to follow a script (as done by other WOZ experiments, e.g. (Prommer et al, 2006; Türck, 2001)),

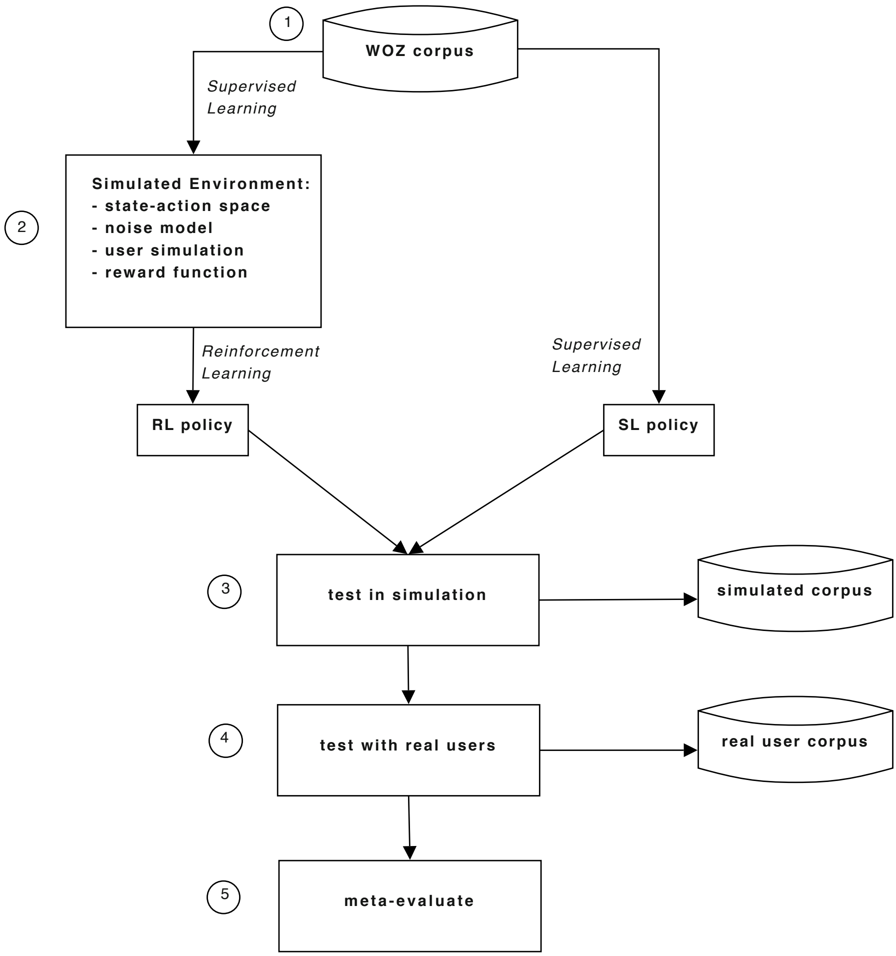

Fig. 5.2 Data-driven methodology for simulation-based dialogue strategy learning for new applications

but the wizard can interact freely with the user. Based on these initial insights about human behaviour we define the state and action set for learning, following Williams and Young (2004b).

These initial strategies should also account for noise in the interaction, as they should be suited to be applied in spoken dialogue systems (which are much more sensitive to noise). Error or uncertainty handling strategies (such as implicit or explicitly confirming what was said) are much less frequent in standard human-human communication, as we previously found in a comparative corpus study of human-human interaction (Rieser and Moore, 2005). We therefore artificially introduce simulated noise by randomly deleting words in the user transcripts before presenting them to the wizard.

The application domain of the experiment is multimodal information-seeking dialogue with an in-car music-player (see Section 3.4). In total we collected data from 21 sessions, containing 72 dialogues with approximately 1600 turns in this setup. A detailed overview of the experiment follows in Chapter 6 (also see (Rieser et al, 2005)). Note that the experiments were conducted in the larger context of the TALK project.[3] The resulting corpus is known as the SAMMIE-2[4] corpus (also see (Kruijff-Korbayová et al, 2005a,b, 2006a,b)).

5.3.2 Step 2: Build a Simulated Learning Environment

In the second step we use the WOZ data to construct a simulated learning environment. In particular, we model the following components: The action set and state space for learning, the user and noise simulation, and the objective function for learning and evaluation. All of these components are constructed using data-driven methods. The action set and state space are retrieved by exploring the wizards' actions. The user and noise simulations are both constructed using frequency-based approaches. The objective function is a predictive model of user ratings obtained by a regression analysis, following the PARADISE framework (see Section 2.2.2). Here one of the major questions is how to construct simulations from small amounts of data, as discussed before in Section 5.2.1. onstruct models with full More detail will be provided in Chapter 7 (also see (Rieser and Lemon, 2006a,b,c, 2011)).

5.3.3 Step 3: Train and test a strategy in simulation

In the third step of our framework, we train and test a policy by interacting with the simulated environment using Reinforcement Learning. We formulate the problem as a hierarchical Markov Decision Process (MDP) which reflects the structure of information-seeking dialogues (i.e. the information acquisition and the subsequent presentation phase). For strategy training we use the SHARSHA algorithm with linear function approximation. We test the RL-based strategy against a baseline which reflects the human wizard behaviour as observed in the data. This baseline is constructed by using Supervised Learning.

Note that RL is fundamentally different to Supervised Learning (SL): RL is a statistical planning approach which allows us to find an optimal policy (sequences of actions) with respect to an overall goal (Sutton and Barto, 1998); SL in contrast

[3] TALK (Talk and Look: Tools for Ambient Linguistic Knowledge; `www.talk-project.org` (20. September 2011)) was funded by the EU as project No. IST-507802 within the 6th Framework program.

[4] SAMMIE stands for Saarbrücken Multimodal MP3 Player Interaction Experiment.

is concerned with deducing a function from training data for predicting/classifying events. This book is not concerned with showing differences between SL and RL on a small amount of data, but we use SL methods to capture the average human wizard strategy in the original data, and show that simulation-based RL is able to find new policies, which were previously unseen. The supervised wizard strategy will be explained in Section 7.6 in more detail. The simulated-based learning experiments will be presented in Section 7.11 (also see (Rieser and Lemon, 2008d)).

5.3.4 Step 4: Test with Real Users

In the fourth step of our framework the strategy is tested with real users. We therefore develop a music-player dialogue system using a rapid development tool. The learned strategy is implemented using a table look-up between states and learned actions. In the user tests 17 subjects interact with the system, solving 2×6 tasks with each policy (SL and RL). At the end of each task they also fill out a questionnaire. The experiments will further be described in Chapter 8 (also see (Rieser and Lemon, 2008c, 2011)).

5.3.5 Step 5: Post-Evaluation

In the final step of our framework we address the problem that a WOZ experiment itself is only a simulation of real HCI. We therefore compare different aspects of the 3 corpora gathered so far: the WOZ study, the dialogues generated in simulation, and the final user tests. We first compare dialogue strategy performance obtained in simulation with the results obtained when testing with real users. We also compare the experimental conditions of the different studies, where we discuss the noise model in particular. We furthermore explore whether the objective function used for learning is a realistic estimate of real user preferences. We will provide more detail in Chapter 8.5 (also see (Rieser and Lemon, 2008a, 2011)).

5.4 Summary

This Chapter introduced and motivated a 5-step procedure for bootstrapping an optimised strategy from WOZ data using Reinforcement Learning methods. We start with a data collection in a modified WOZ setup. In this setup the wizard also becomes a subject and channel noise is artificially introduced. We then construct a simulated learning environment from this data, where all the simulated components are constructed using data-driven methods suited for learning from small data sets. The strategy is then trained and tested by interacting with this simulated environment.

We compare RL against a supervised baseline which is derived from the wizards' behaviour. This comparison allows us to measure relative improvements over the training data. We also evaluate the strategy with real users (see Chapter 8). Finally, we post-evaluate the policy and the simulated environment by comparing the WOZ data, the simulated data, and the real user corpus. We also conduct a detailed error analysis, see Section 8.5.

Annotated example dialogues from the 3 different corpora can be found in Appendix A: Appendix A.1 contains dialogues from the WOZ study, Appendix A.2 simulated dialogues, and Appendix A.3 examples from the user study.[5]

We now apply this framework to optimise multimodal information-seeking dialogue strategies for an in-car digital music player. Dialogue Management and multimodal output generation are two closely interrelated problems for information seeking dialogues: the decision of *when* to present information depends on *how many* pieces of information to present and the available options for *how* to present them, and vice versa. We therefore formulate the problem as a hierarchy of joint learning decisions which are optimised together. We see this as a first step towards an integrated statistical model of Dialogue Management and more advanced output planning/Natural Language Generation, see Chapter 9.

[5] Note that we will refer to these example dialogues in different chapters throughout this book . We therefore decided to put the example dialogues in the Appendix in order to facilitate consistent reference and allow the reader to directly compare dialogues from different corpora more easily.

Chapter 6
Data Collection in a Wizard-of-Oz Experiment

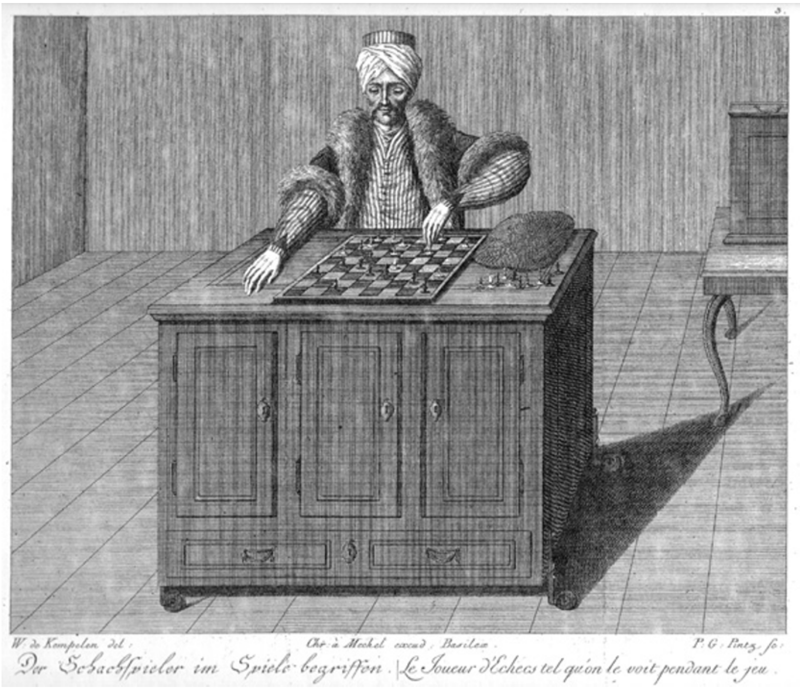

Fig. 6.1 An engraving of Kempelen's chess "automaton" *The Turk* from Karl Gottlieb von Windisch's 1784 book "Inanimate Reason"

It is a common approach within the area of Artificial Intelligence to have a human being simulate the intelligent behaviour of a machine, before one is able to build a full working system. This method is used for different reasons. Either one needs to collect initial data before a system is designed, or one needs to produce more intel-

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_6,

ligent behaviour than is currently possible by machines. One example of the latter is *The Turk*, see Figure 6, constructed and unveiled in 1770 by Wolfgang von Kempelen (1734 - 1804) to impress the Empress Maria Theresa of Austria-Hungaria. Note that, Wolfgang von Kempelen also was the first to invent Speech Synthesis (see Chapter 2). Publicly promoted as an 'intelligent' automaton, the Turk was in fact a mechanical illusion that allowed a human chess master to hide inside and operate the machine. With a skilled operator, the Turk won most of the games played (Standage, 2002).

In this Chapter we simulate interaction between a human and a machine for the purpose of data collection. In particular, we simulate real Human-Computer Interaction (HCI) in a Wizard-of-Oz (WOZ) experiment (Dahlbäck et al, 1993; Fraser and Gilbert, 1991), which follows a similar idea to the Turk. In a WOZ experiment a human operator, the so-called "wizard", simulates (partially or completely) the behaviour of a dialogue system, while the user is lead to believe that s/he is operating with a real system (see Section 3.3.1). Within the context of the book we use WOZ data to 'bootstrap' an optimised dialogue strategy using Reinforcement Learning (RL). Using WOZ data, as opposed to data from real HCI, also allows us to study the behaviour of the human operator before specifying the system's functionality. We therefore make several changes to the conventional WOZ setup (which is described in Section 3.3.1). First of all, we are interested in the actions that a human wizard intuitively employs in a specific context. We therefore do not restrict the wizards by using a script (as done by other WOZ experiments, e.g. (Prommer et al, 2006; Türck, 2001)), but the wizards can interact freely with the user (see Section 6.1). In addition, the wizard only "sees what the system sees", i.e. features which are available for decision making at system runtime. In particular we simulate channel noise, as introduced by Automatic Speech Recognition, as described in Section 6.2. Furthermore, we convert the data into a special data format, which facilitates analysis and annotation, as described in Section 6.3. Finally, we analyse the quality of the employed strategies, as described in Section 6.4.

The experiments described in the following chapter were conducted in the larger context of the TALK project.[1] The resulting corpus is known as the SAMMIE-2[2] corpus (also see (Kruijff-Korbayová et al, 2005a,b, 2006a,b)).

6.1 Experimental Setup

The experimental setup for a WOZ study should resemble the intended application domain as closely as possible. In our case, we simulate a multimodal in-car dialogue systems for interactive search in a music database (see Section 3.4.3). In the following we describe some of the details of the experiment. Different aspects of

[1] TALK (Talk and Look: Tools for Ambient Linguistic Knowledge; `www.talk-project.org` (20. September 2011)) was funded by the EU as project No. IST-507802 within the 6th Framework program.

[2] SAMMIE stands for Saarbrücken Multimodal MP3 Player Interaction Experiment.

the experiment are also described in Becker et al (2004); Korthauer et al (2006); Kruijff-Korbayová et al (2005a); Rieser et al (2005).

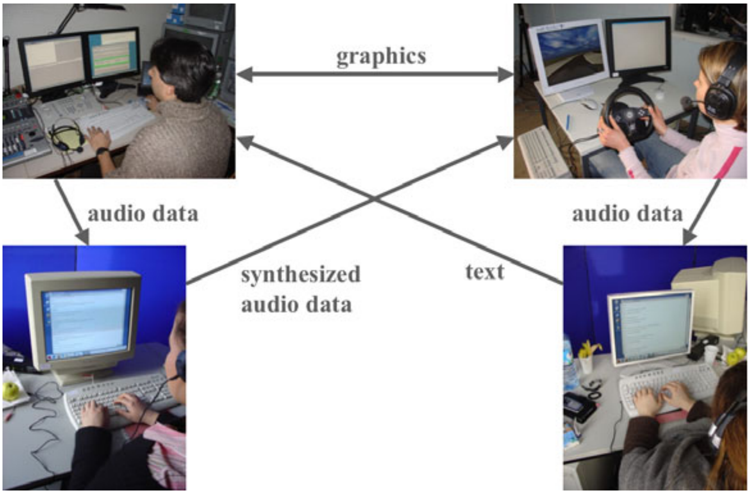

Fig. 6.2 Multimodal Wizard-of-Oz data collection setup for an in-car music player application, using the Lane Change driving simulator. Top right: User, Top left: Wizard, Bottom: transcribers

The experimental setup is shown schematically in Figure 6.2. There are five people involved in each session of the experiment: an experiment leader (not shown), two transcribers, a user, and a wizard. The interactions were carried out in German.[3] The wizards play the role of an intelligent interface to a music player and are given access to a database of music information (but not actual music) of more than 150,000 albums (almost 1 million songs), extracted from the FreeDB database.[4] Figure 6.3 shows an example screen shot of the music database as it is presented to the wizard. The database also returns partial matches, e.g. if the wizard searches for 'Subterranean Homesick' the song 'Subterranean Homesick Alien' by Radiohead as well as the the song 'Subterranean Homesick Blues' by Bob Dylan are retrieved.

The users are given a set of predefined tasks and are asked to accomplish them by using a music player with a multimodal interface (as further described in Section 6.1.2). In a part of the session the users are also given a primary driving task, using the *Lane Change* driving simulator (Mattes, 2003) in order to resemble the in-car eyes-busy driving situation (see Section 3.4.3). The wizards can speak freely and display the search results or playlists on the screen. The users can also speak, as well as making selections on the screen. The user's utterances are immediately transcribed by a typist and are also recorded. The transcription is then presented to the wizard. This was done in order to deprive the wizards of information encoded in the

[3] However, most of the titles and artist names in the music database are in English.

[4] Freely available at `http://www.freedb.org/` ((20. September 2011))

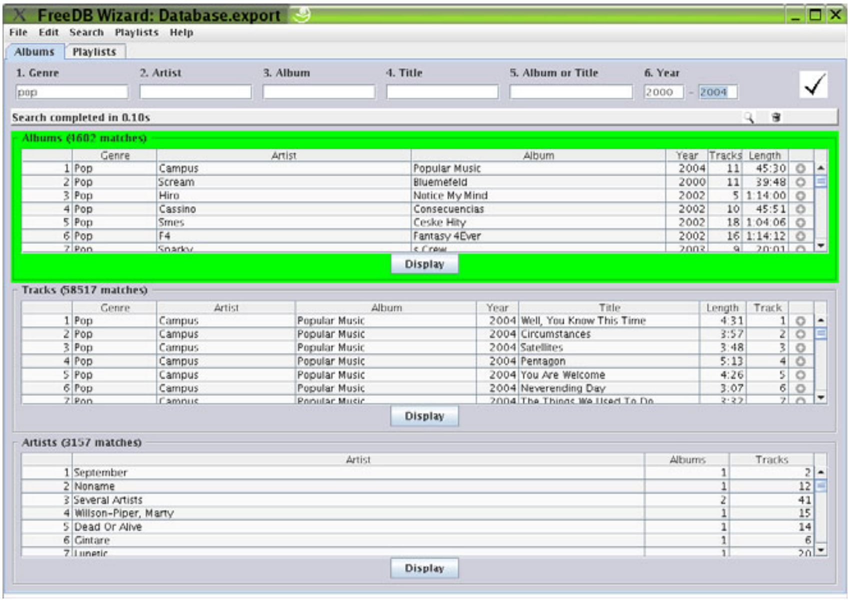

Fig. 6.3 Screenshot from the FreeDB-based database application, as seen by the wizard

intonation of utterances (as in current dialogue systems), and in order to be able to corrupt the user input in a controlled way, simulating understanding problems at the acoustic level (as further described in Section 6.2). The wizard's utterances are also transcribed (and recorded) and presented to the user using the speech synthesiser system Mary.[5] The transcription is also supported by a typing and spelling correction module to minimise speech synthesis errors and thus help maintain the illusion of a working system.

Since it would be impossible for the wizard to construct layouts for screen output on the fly, s/he gets support for this task from the WOZ system: The system automatically computes four possible screens, as shown in Figure 6.4. The wizard can chose one of the offered options to display to the user, or can decide to clear the user's screen. Otherwise, the user's screen remains unchanged. It is therefore up to the wizard to decide whether to use speech only, display only, or to combine speech and display.

The types of screen output are (i) a simple text-message conveying how many results were found, (ii) output of a list of just the names (of albums, songs or artists) with the corresponding number of matches (for songs) or length (for albums),(iii) a table of the complete search results, and (iv) a table of the complete search results, but only displaying a subset of columns. These four screens are presented to the

[5] The Mary system for German TTS is available at `http://mary.dfki.de/` (3. January 2011)

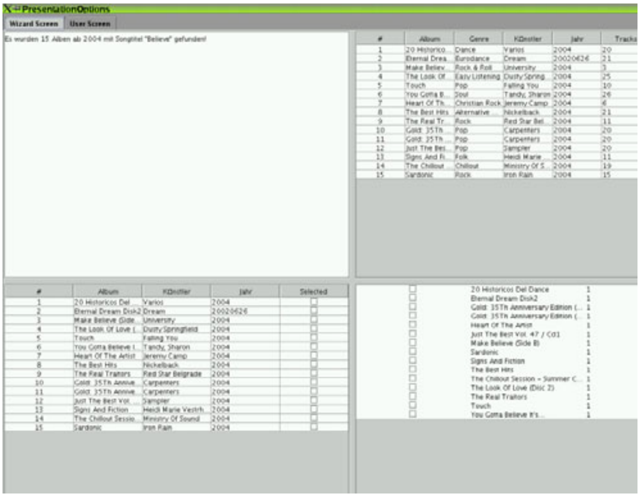

Fig. 6.4 Screenshot from the display presentation tool offering options for screen output to the wizard for second-level of choice what to display an how

wizard in different quadrants on a monitor (cf. Figure 6.4), allowing for selection with a simple mouse click.

This setup is technically realised using the Open Agent Architecture (OAA) (Cheyer and Martin, 2001), which is a framework for integrating a community of software agents in a distributed environment. Each system module is encapsulated by an OAA wrapper to form an OAA agent, which is able to communicate with the OAA community. The experimental system consists of 12 agents (all of them written in Java).

6.1.1 Recruited Subjects: Wizards and Users

In our study we are interested in the behaviour of two types of subjects: users and wizards. We recruited 5 wizards (2 female, 3 male) [6] in total, all were native speakers of German with good English skills (in order to understand and search for English titles), all were between 20 and 35 years old, and all had no prior experience with language technology or dialogue systems. Each wizard contributed about equal portions to the corpus (16.4 – 25.4%).

In addition we recruited 21 user subjects (11 female, 10 male). Most of them are between 20-30 years old (66.7%) and their field of study is spread between subject areas (social science: 23.8%, languages: 23.8%, natural science: 28.6%, arts:

[6] Originally we had 6 wizards performing the task. Due to data loss caused by technical failure, complete data only exists for 5 of our 6 wizards.

17.1%). Each subject received a compensation of 15.- €. The wizards were paid per hour.

6.1.2 Experimental Procedure and Task Design

The experiment proceeded as follows. First the wizards were trained to use the database interface and they were also given general instructions about how to interact with the user. For example, that they should avoid relying on their own music knowledge, but should instead ask the user for missing values. Training took 45 minutes, including 5 example tasks. User and wizard were placed in separate rooms. When the user arrived s/he also first received a sheet of instructions to read. After the user arrived s/he was introduced to the driving simulator and had to perform a short test drive. The users solved two sets of tasks with two tasks in each. The task description is handed out on separate board cards, one for each task, so that the user can decide to solve the tasks in any preferred order. After each task the user filled out a task-specific questionnaire, where the they indicated perceived task success and satisfaction on a 5-point Likert Scale. Finally, the user was interviewed by the experiment leader following a questionnaire containing similar questions to the PARADISE study (Walker et al, 2000), including questions on task ease, timing, multimodal and verbal presentation, as well as future use of such systems. Details of the questionnaire are provided in Section 6.4.2.

We designed 10 task sets in total. Every task set was used at least twice, and each wizard never used the same task twice. Each task set contains 4 tasks of 2 different types. One task type asks the user to search for a specific title or album, the other one asks the user to build a playlist which has to satisfy a number of constraints. The tasks are paired into 2 compatible sets, containing one task of each task type.

6.2 Noise Simulation

6.2.1 Related Work

One major difference between real HCI and WOZ experiments is that the wizard usually has full understanding of the user's utterance, whereas the interaction with a real dialogue system is affected by channel noise (see Section 3.3.1). Hence, recent work has suggested to introduce artificial noise in the WOZ setup in order to get a closer estimate of real HCI.

Skantze (2003, 2005) was the first to introduce channel noise in a WOZ setup. He uses a real ASR system to study different error handling strategies as applied by the wizards. Stuttle et al (2004); Williams and Young (2004a) follow this method, but simulate ASR errors in order to control the error rate and thus to be able to investigate how the wizards adapt to different levels of noise. Results by Williams and Young (2004a) show that even with high WER conditions the wizards are able to parse ASR output well and assimilate contextual knowledge about what user actions are likely to follow. Such abilities are currently unrealistic for spoken dialogue systems (but see (Gabsdil and Lemon, 2004; Lemon and Konstas, 2009) who use context to predict likely next user actions and thereby improve ASR). In order to prevent the wizard from using too much contextual knowledge, we deleted parts of the user's input completely, as described below.

6.2.2 Method

To approximate speech recognition errors we used a tool that randomly "deletes" parts of the transcribed utterances. Due to the fact that humans are able to make sense of even heavily corrupted input, this method not only covers non-understandings, but wizards also built up their own hypotheses about what the user really said, which can lead to misunderstandings. The word deletion rate varied: 20% of the utterances were weakly corrupted (= deletion rate of 20%), and 20% were strongly corrupted (= deletion rate of 50%). In 60% of the cases the wizard saw the transcribed speech uncorrupted. Example 6.2.1 illustrates the kind of corrupted utterances the wizard had to deal with.

Example 6.2.1
uncorrupted: *Zu dieser Liste bitte Track 'Tonight' hinzufügen.*
[*Add track 'Tonight' to this list.*]
weakly corrupted: *Zu dieser Liste bitte Track Tonight ....*
[*... track 'Tonight' to this list.*]
strongly corrupted: *Zu ... Track Tonight ....*
[*... track 'Tonight' to ....*]

6.2.3 Results and Discussion

Altogether, only 30% of the corrupted utterances had a noticeable effect on the interaction, i.e. the wizard did indicate some problem with understanding what was said (see corpus-based example in Appendix A.1, Table A.2). The non-understanding and misunderstandings are manually annotated as described in Blaylock et al (2006). In total 7% of all the user turns lead to a communication error, which is much lower than the current WER for spoken dialogue systems, where error rates can be around 30% (see Chapter 7.7). On the other hand, the error rate is higher than for human-human communication. For example, when people talk over a telephone line only 1.5% of all utterances are followed by an acoustic communication error (Rieser and Moore, 2005).

There are some shortcomings of this technique, as also pointed out by Schlangen and Fernandez (2007), who used a similar setup to simulate a noisy communication channel. Deleting words is a rather crude simulation of real-world acoustic problems. Note that there are also studies introducing errors from ASR (Skantze, 2005; Stuttle et al, 2004). We think however, that deleting words is simulating more "natural" communication problems (e.g. some parts of an utterance might be distorted by transient noise). It is not clear whether confronting human wizards with actual ASR errors will reveal the range of natural behaviour we are interested in. In contrast to real ASR errors, our deletion method has the potential to elicit more 'natural' behaviour from the wizards. Complete deletions are similar to transient noise, whereas substitutions and insertions as introduced by ASR (see Chapter 7.7) normally won't happen in human-human communication. Therefore we believe that our method is better suited to study natural approaches to dealing with noise.

A major drawback of our method is the time delay introduced by transcribing the utterances (the user as well as the wizard utterance). On average, the user has to wait $6.7(\pm 5.2)$ seconds for the wizard to reply. Most of the wizards reported that they adjusted their behaviour accordingly. For example, they would omit some questions in order to speed up the dialogue. On the other hand, it seemed that the slow interaction pace made the users believe that they were interacting with a real system.[7]

An alternative method for inserting deletions was later introduced by Schlangen and Fernandez (2007). Schlangen and Fernandez distort the acoustic signal by using white noise. While this method is less time (and cost) intensive, they do not prevent the wizard from using intonation – a feature which usually is not accessible for real HCI. Schlangen and Fernandez report similar results, where 71% of noise events did not have an noticeable effect.

In sum, our results indicate that the applied method still does not prevent the wizards from using contextual knowledge to parse corrupted user input. Thus, too many utterances are correctly understood compared to real HCI. As such, this method is not suited to study detailed error handling. However, this method can be sufficient in order to study natural presentation strategies under the presence of noise, as we show in Chapters 7,8, and 8.5.

6.3 Corpus Description

In total we gathered 21 sessions, containing 72 dialogues with about 1600 turns. Some example dialogues can be found in Appendix A.1. The data for each session consists of a video and audio recording, the questionnaire data, transcripts, and a log file. The gathered logging information per session consists of OAA messages in chronological order, each marked by a timestamp. The log files contain various information, e.g. the transcriptions of the spoken utterances, the wizard's database

[7] Unfortunately we failed to include a question in the questionnaire asking the user explicitly whether s/he believed that she interacted with a real system.

query and the number of results, the screen option chosen by the wizard, and so on. In addition, the speech data is manually transcribed using the Transcriber Tool.[8]

The corpus is marked-up and annotated using the Nite XML Toolkit (NXT) (Carletta et al, 2005). The NXT format allows multi-layered annotation, where each layer is annotated independently, but subsequent investigations can involve exploration and automatic processing of the integrated data across layers. We created the NXT-based corpus in several steps: First we automatically converted the transcriptions and information from the log files. Then, we added various features using a mixture of manual and (semi-)automatic annotation techniques (see (Kruijff-Korbayová et al, 2006a,b)). For example, we manually annotated Clarification Requests (CRs), as described in Chapter 7.3. We therefore created several annotation tools, based on the NXT core library to facilitate complex manual annotation tasks (see Figure 6.3 for an example annotation tool created to annotate Clarification Requests).

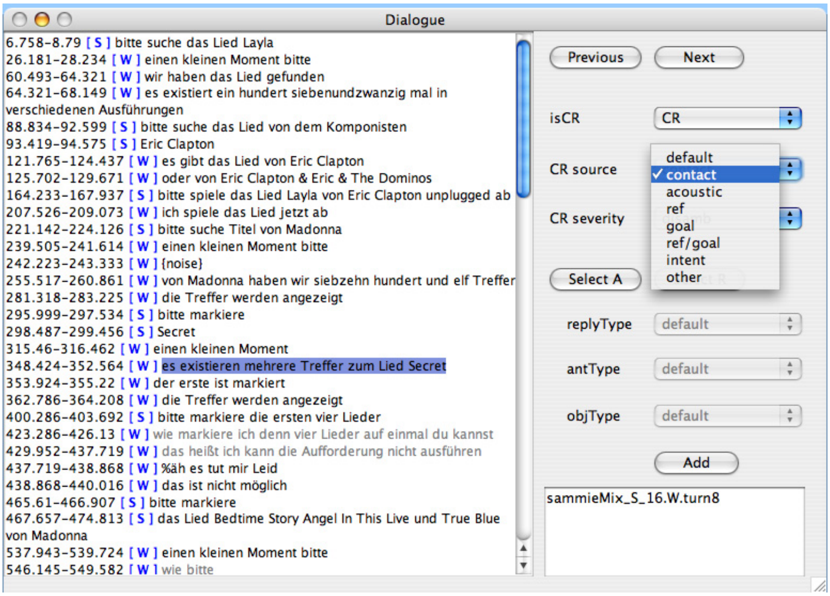

Fig. 6.5 NXT-based tool for annotating Clarification Requests

In addition, we also manually annotated task success as described in (Blaylock et al, 2006). Task success is calculated as the ratio of information slots defined by the task compared to the information slots actually filled and confirmed during the interaction. Furthermore, we annotated many contextual dialogue features, as further described in Section 7.4. Many of them are automatically annotated using NXT indexing (Carletta et al, 2005).

[8] `http://trans.sourceforge.net/` (22. April 2008)

6.4 Analysis

We now present the results of the corpus analysis for multimodal presentation strategies. We first summarise qualitative measures of the wizards' and users' behaviour. We then summarise the results from the user questionnaire.

6.4.1 Qualitative Measures

Here we describe the multimodal behaviour of the dialogue participants in more detail. A detailed analysis of the frequencies of annotated wizards' actions can be found in Section 7.3. An analysis of the annotated user's actions can be found in Section 7.8. In this Section we are especially interested in the presentation strategies applied by the wizards.

Of the 793 wizard turns 22.3% were annotated as presentation strategies, resulting in 177 instances for learning, where the six wizards contributed about equal proportions.Overall, the wizards choose to present a screen output in 48.0% of all the wizard turns. The table option is presented most frequently (78.6%). The option to present a list is chosen less frequently (17.5%), and the option to present text only is hardly ever used (0.04%). The presented lists and tables contain 167.9($\pm$177.7) items on average, with the maximum number of 1711 items (see Example 6.4.1) and the minimum of 1 item displayed on the screen. For multimodal presentation the wizards commonly name the number of results while displaying the chosen screen output, e.g. "The song is retrieved. There are 142 different versions." (see Appendix A.1, Table A.1).[9]

Example 6.4.1
User: *Please search for music by Madonna .*
Wizard: *I found seventeen hundred and eleven items. The items are displayed on the screen.* [displays list]
User: *Please select 'Secret'.*

For verbal presentation the wizards only present 1.6($\pm$.45) items on average, with a maximum of 3 items verbally enumerated. A common pattern observed for verbal presentation is that the wizard summarised the results by presenting the options for the most distinctive feature to the user, e.g. " The song exists by 'Eric Clapton' or by 'Derek & the Dominos'." (see Appendix A.1, Table A.1).

Overall, the tasks are solved with 87.6%($\pm$25.14) task success. One dialogue takes on average 14.5($\pm$8.8) turns. A turn is defined is defined as in (Paek and Chickering, 2005): it begins at the start of each new user action (since all system actions respond to user actions). A user action can be an utterance, or a click (i.e. the user selected an item on the screen).

[9] Note that the database used in the WOZ experiments contained over a million songs and is searched with partial string match. Therefore the retrieved items for 'Layla' can be as high as 142.

As we want to use the data for learning, it is important to know whether the individual wizards applied significantly different strategies. We therefore apply a one-way ANOVA followed by a planned comparison in order to compare the mean value of different dialogue performance features across wizards. Dialogue length only varies slightly across wizards ($F(4) = 2.84, p = .04$). Wizards also present about the same amount of information on the screen ($F(4) = .76, p = .55$). Most of the wizards are equally successful in completing the tasks. Only wizard 4 was significantly better ($p < .001$) than wizard 2, as wizard 4 completes all the tasks with 100% task success whereas wizard 2 only completes the tasks with 78.4% success rate. Overall, one can say that all the wizards apply similar strategies with respect to average measures. Note, that this does not mean that wizards reacted the same way in specific dialogue contexts.

The users' multimodal behaviour is very limited. Only 3 users selected an item on the screen by clicking. We only observe 11 clicking events in total. However, the user verbally referred to the displayed list on average $0.5(\pm.96)$ times per dialogue.

6.4.2 Subjective Ratings from the User Questionnaires

At the end of the interaction the user is interviewed using a questionnaire.The questionnaire contains 28 questions in total, many of them targeted to different research questions within the TALK project. We are interested in the questions targeting User Satisfaction as defined by PARADISE (see Section 2.2.2) and the multimodal presentation strategy. All subjects reported that they were convinced that they were interacting with a real system.

The results for the mean PARADISE user ratings are displayed in Figure 6.6. ASR performance is rated highest with $3.9(\pm.4)$ points on a 5-point Likert Scale. Expected Behaviour and Interaction Pace are rated worst with $1.8(\pm.6)$ due to the time delay introduced by transcribing. The other ratings are between 2 and 3 points which is about average on the 5-point scale. In order to compare the performance to other systems we calculated the overall user satisfaction as the sum of five core metrics (TTS performance, Task Ease, User Expertise, Expected Behaviour, and Future Use) following
citetwalker:csl02.[10] The mean for user satisfaction across all dialogues is $15(\pm2.9)$. Walker et al report an average user satisfaction of 16.2 for 9 Communicator systems. These results indicate that the performance of our human wizards is perceived to have a comparable quality to previous dialogue systems.

To measure task experience we elicit data on perceived task success and satisfaction on a 5-point Likert scale after each task was completed (as described in Section 6.1.2). Subjects rate both measures above average: the mean perceived task success

[10] Note that
citetsassi criticise that taking the sum of different dimensions produces meaningless results. We therefore only use this measure to compare our results to other systems, and do not engage in the discussion of the interpretation of this score.

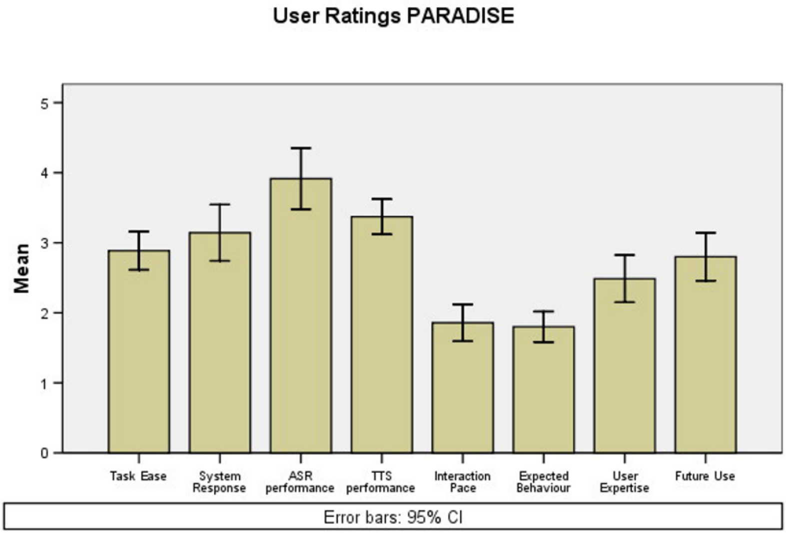

Fig. 6.6 User Ratings on the PARADISE questions on a 5-point Likert Scale

is $3.4(\pm.8)$ and the mean perceived task satisfaction is $3.8(\pm.8)$. These high ratings indicate that the WOZ 'system' is perceived to be very usable in general, and the users are under the impression that they are able to accomplish most of their tasks.

The mean user ratings for the multimodal presentation strategy are shown in Figure 6.7. Multi-modal presentation strategies are perceived to be helpful in general, having a mean of $3.1(\pm1.7)$ for switching between different modes within one dialogue and a mean of $2.9(\pm1.2)$ for combining different modes within one dialogue on a 5-point Likert scale.

However, the subjects report that too much information is being displayed, especially for the tasks with driving. Users also find that the interaction distracted them from driving (see Figure 6.7 'cognitive load', note that the scale is reversed). 85.7% of the subjects report that the screen output (sometimes) distracted them. The amount of information presented on the screen is rated low with a mean of $2.2(\pm.8)$ (see Figure 6.7, 'multimodal info'). 76.2% of the subjects would prefer to more verbal feedback, especially while driving. The amount of information presented verbally is rated as being appropriate (mean of 2.9 ± 1.0).

In sum, the user ratings indicate that multimodal presentation was only perceived to be appropriate in particular *contexts*. To further explore this hypothesis we conducted a χ^2 test on presentation strategy (comparing whether wizards chose to present in multimodal or verbal modality). The results showed significant differences between wizards ($\chi^2(1) = 34.21, p < .001$). On the other hand, a Kruskal-Wallis test comparing user preferences for the multimodal output showed no sig-

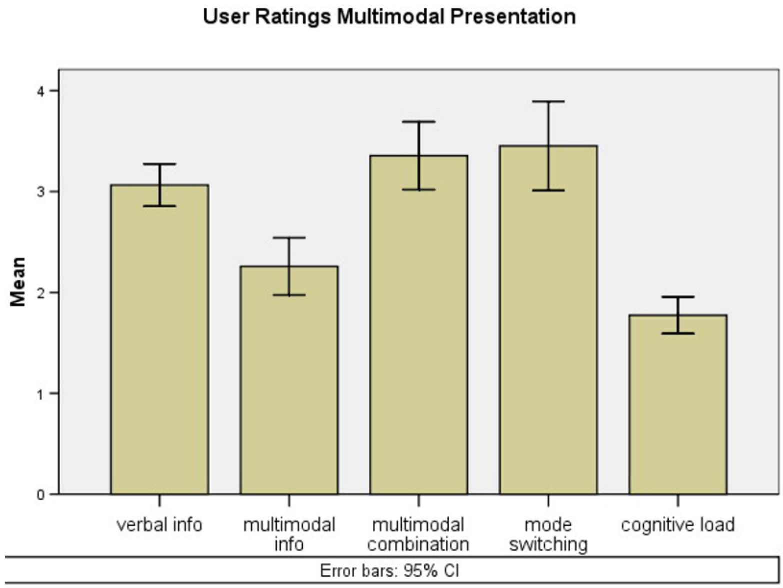

Fig. 6.7 User Ratings on the multimodal presentation questions on a 5-point Likert Scale

nificant difference across wizards ($H(5)=10.94$, $p > .05$). [11] Mean performance ratings for the wizards' multimodal behaviour ranged from 1.67 to 3.5 on a five-point Likert scale. We also performed an analysis of whether wizards improved their performance over time (learning effects). However, the results show that the wizard's average user satisfaction scores in general slightly decreased with the number of sessions that they performed.

Observing significantly different strategies which are not significantly different in terms of user satisfaction, we conjecture that the wizards converged on strategies which were appropriate in certain *contexts*. To strengthen this hypothesis we split the data by wizard and performed a Kruskal-Wallis test on multimodal behaviour per session. Only the two wizards with the lowest performance score showed no significant variation across session, whereas the wizards with the highest scores showed the most varying behaviour. These results again indicate a context-dependent strategy.

The dialogues show that common "mistakes" were that the wizards either displayed too much information on the screen, see Example A.1 in Appendix A.1. or that the wizards fail to present results early enough, see Example A.2 in A.1. Screen outputs should only contain an 'appropriate' amount of information. In other words,

[11] The Kruskal-Wallis test is the non-parametric equivalent of a one-way ANOVA. Since the users indicated their satisfaction on a 5-point Likert scale, an ANOVA which assumes normality would be invalid.

we need to find a strategy which decides on *how many* database search result to present to the user, and *when* to present them, and *which modality* to use for presentation, in an adaptive, context-dependent, 'optimal' manner.

These results indicate that we need to find a strategy given the competing trade-offs between the number of results (large lists are difficult for users to process), the length of the dialogue (long dialogues are tiring, but collecting more information can result in more precise results), and the noise in the speech recognition environment (in high noise conditions accurate information is difficult to obtain).

In the next chapter we use the ratings from the user questionnaires to optimise a presentation strategy using simulation-based RL.

6.5 Summary and Discussion

The analysis of the WOZ data shows that the users perceive the wizards' behaviour to be sub-optimal. In particular, the amount of multimodal information as displayed on the screen is often perceived to be inappropriately large. For example, the wizards choose to show up to 1711 items on the screen. On the other hand, users prefer strategies which are short and efficient. We therefore need to find a policy which addresses the problem of *how many* database search results to present to the user, and *when* to present them, and *which modality* to use for presentation, given the competing trade-offs between the number of results (large lists are difficult for users to process especially when only presented verbally), the length of the dialogue (long dialogues are tiring, but collecting more information can result in more precise results). In order to obtain this policy we will optimise presentation strategies with respect to user preferences, using simulation-based Reinforcement Learning, as described in the next Chapter.

Given our small data set, we apply the following simplification for strategy learning: We do not distinguish between different types of screen outputs (i.e. the 4 output options available to the wizards cf. Figure 6.4), but only use the table as the only multimodal option available (since this was the one chosen most frequently by the wizards). For the learning experiments we simplify the decision to verbal versus multimodal presentation. The option that the wizard only presents a screen output without verbal commentary did not occur in the data (as we will show in Section 7.3).

A shortcoming of the WOZ setup described in this Chapter is that the questionnaire is only taken at the very end of the experiment. User ratings only exist for the whole interaction, including driving and non-driving scenarios. Thus, the obtained user ratings do not distinguish between the condition for driving and not driving. On the other hand, it seems to be the case that wizards seem not to pay attention to the fact whether the user is driving when choosing their presentation strategy. Only 2 of our 5 wizards reported that they (sometimes) took into account whether the user was driving or not when presenting search results.

Another shortcoming is that we do not log information about the users' driving performance. Other work chooses 'cognitive load' as the measure to be minimised by in-car dialogue strategies, e.g. (Kun et al, 2007; Wada et al, 2001). We consider learning presentation strategies which optimise driving performance as an interesting direction for future work.

Another direction for future work is to create a WOZ setup which is more realistic with respect to the final application. In this WOZ study the components for multimodal presentation do not resemble a real car environment. For example, the screen used to display the results has the size of a standard computer screen, whereas for an in-car dialogue the available screen size usually is much smaller. Furthermore, in the current experiments the user has to select the items with a mouse click, which is rather difficult when driving (in fact, only 3 users used the mouse, see Section 6.4.1).

In the next Chapter we use the WOZ data to 'bootstrap' different aspects of the simulated learning environment for RL-based strategy development.

Chapter 7
Building Simulation Environments from Wizard-of-Oz Data

Fig. 7.1 Wooden mechanical horse simulator during World War I – Learning with simulated environments

This Chapter describes a framework for training and testing a RL-based policy in simulation, where the simulated environment is obtained from limited amounts of Wizard-of-Oz (WOZ) data.

7.1 Dialogue Strategy Learning with Simulated Environments

Why do we need simulated interaction?

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_7,

In general, a simulation is an imitation of some real thing, state of affairs, or process. Learning in simulation allows the exploration of various strategies in a "low cost" environment where trying alternatives has a low risk. For example, during World War I soldiers were trained using a physical simulation of a horse, see Figure 7; nowadays soldiers are trained in virtual environments, e.g. (Traum et al, 2008).

The same is true for dialogue strategy optimisation using Reinforcement Learning (RL). Instead of having real users interacting with the system we let the system "practise" in a "low risk" simulated environment until it is skilful enough to interact with real users. RL-based strategy design is based on exploratory trial-and-error learning methods (see Section 3.2). In exploratory trial-and-error learning the learning agent may also explore polices which do not make sense to real users. Thus, exploratory trial-and-error learning with real users is a time-consuming and potentially sadistic procedure, where real users can easily get irritated and frustrated. (For further discussion on advantages and disadvantages of simulation-based RL please see Section 3.2.3.)

The quality of the learned strategy depends on the quality of the simulation in the sense that the strategy learns by interaction with the simulated environment, as results by Schatzmann et al (2005b) indicate. Thus, building (and evaluating) simulated environments for dialogue policy learning has become a research area in its own right, e.g. (Ai et al, 2007b; Georgila et al, 2006b; Pietquin and Dutoit, 2006b; Pietquin and Hastie, 2011; Rieser and Lemon, 2006a; Schatzmann et al, 2006). Some of the research issues for simulation-based dialogue strategy learning are summarised in the following.

First of all, there is no agreed definition of what a "simulated environment" constitutes. For example, it is not clear where to exactly draw the boundary between the RL agent and the learning environment. Sutton and Barto (1998) give the general guideline that "anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment" (Sutton and Barto, 1998, p.53). Given this definition we take everything but the learning mechanism itself to be part of the environment.

Furthermore, there is no general agreement on which components to include in the simulated environment. At the very least, a simulated user is needed for simulation-based dialogue strategy learning. Some work also includes a noise simulation which models errors as introduced by ASR, e.g. (Pietquin and Dutoit, 2006b), while other work has ignored the error channel altogether, e.g. (Levin et al, 2000; Schatzmann et al, 2007c). In general, which dialogue components are required in the simulation depends on the application domain of the strategy and the research questions to be addressed.

In addition, it is also not clear how to obtain these simulated learning environments in general. Most of previous work uses a mixture between components which are data driven (in particular the user simulation is often obtained from data), and components which are handcrafted (in particular the reward function often is set manually). Different methods are used for different components and we will discuss them individually in this Chapter. The data-driven methods used in general fall into the category of Supervised Learning (SL). Handcrafting of the simulated components is not a useful method, since the resulting components are rather unlikely to reflect the application domain correctly, as discussed in Section **??**. In the following we propose what is currently the most strongly data-driven approach to these problems. We suggest to learn supervised models from WOZ data, and we identify the following challenges:

- how to obtain reliable models from a small data set, and
- whether models obtained from simulated HCI are reliable estimates of real dialogue interaction.

We now discuss the question of "how much data is enough?" to obtain simulations for dialogue strategy learning. We then discuss different techniques used to evaluate simulated components. We also show how available corpus size, simulation technique, and evaluation are closely interleaved problems.

7.1.1 Method and Related Work

How much data is enough to build a simulated environment to train RL-based dialogue strategies?

Different techniques have been used to assess the needed data quantity when learning RL strategies from fixed data sets (see Chapter 2.3.4). For example, people report on how many times a state was visited during training, e.g. (Henderson et al, 2008; Singh et al, 2002; Spitters et al, 2007); measure whether the obtained V-values converge (Tetreault and Litman, 2006); or measure the confidence interval for the learned transition function (Tetreault et al, 2007) (also see the discussion in Section 2.3.4).

For simulation-based learning, however, no such measure exists since one assumes that for RL the simulated environment can produce an infinite amount of training data. Here the question is how much data is enough to build simulated components using data-driven techniques. In general, SL is less "data-hungry" than RL, but still some initial data is needed.[1]

Prior work has trained different types of supervised models on corpora of different quantity.

Schatzmann et al (2005b) and Georgila et al (2006b) for example build user simulations from richly annotated dialogues from the COMMUNICATOR corpus (Walker et al, 2002b). Georgila et al (2006b) utilise 1683 annotated dialogues; Schatzmann et al (2005b) utilise a subset of 697 dialogues. They both use different SL techniques (such as n-gram modelling), where they show that more complex models outperform simpler ones. For evaluation they choose different metrics like the (expected) accuracy, recall and precision.

[1] However, for SL "There is no data like more data" (a quote attributed to Bob Mercer, former researcher of the IBM speech group). More data always helps supervised models to become more accurate and robust, as results by Banko and Brill (2001) indicate.

Other work has used much smaller corpora to train their simulated models. Prommer et al (2006), for example, learn from limited data (82 dialogues, 314 utterances). They produce simulations which are "sufficiently accurate". Unfortunately, their simulated learning environment also includes a lot of hand-crafted aspects, and the user models are what they call "low conditioned" as they also contain several states with zero frequencies (as we will further discuss in Section 7.8.1.2).

However, there is also recent work that shows the limitations of learning from small data sets. Results by Ai et al (2007b) indicate that user simulations learned from a small data set (130 dialogues) can fail to provide enough coverage in the strategy training phase. And in fact, random models might even be superior (as we will further discuss in Section 7.8.1).

Taken together, there is no fixed threshold or standard recommendation for how much data is enough. The required data size also depends on the complexity of the problem we try to learn, how representative the data set is of the problem in question, and the required accuracy – a fact which is described as the "triple trade-off problem" for SL (Dietterrich, 2003).

Thus, in order to choose methods for building simulations, we first address the question of the "required accuracy".

A number of different metrics have been suggested for evaluating different simulated components (see Table 7.1 for an overview). In general, the applied metrics assess two main objectives: *accuracy* with respect to the initial data set, or its *utility* for policy learning. (Schatzmann et al, 2006) call this distinction *direct* versus. *indirect* evaluation. However, by measuring the accuracy with respect to the initial data we assume that there is a large and representative enough data set which allows learning of realistic behaviour. Furthermore, measuring the utility has the drawback that we only know about the quality of the simulations after we have trained the policy, and it would be beneficial to have an estimate even before training.

So, what is an objective function suited to building and testing simulated environments?

In Section 5.2.1 we defined the purpose of simulated learning environments to provide realistic feedback to the learner, while covering all the state-action spaces that need to be explored. Thus, random models, as suggested by Ai et al (2007b) are not an option, as they do not give realistic feedback. Models with zero frequencies, as used by Prommer et al (2006) are also not an option as they don't cover all the possible state-action spaces. Furthermore, models which over-fit the data (see definition Section 5.2.1) are also unlikely to produce behaviour which is representative and general enough. Over-fitting is especially problematic when learning from small data sets (Alpaydin, 2004).

Thus, we define our overall objective for building simulated components from small data sets to handle the trade-off between facilitating coverage, while providing realistic feedback, and being general enough not to over-fit the data.

Evaluation	DIRECT	INDIRECT
error model:	• similarity of generated user utterances between simulated and real (initial) corpus (Schatzmann et al, 2007b)	• policy performance in simulation (Pietquin and Beaufort, 2005; Pietquin and Dutoit, 2006a)
User simulation:	• (expected) accuracy, recall, and precision with respect to the user population in the initial data set (Georgila et al, 2006b; Schatzmann et al, 2005a). • perplexity (Georgila et al, 2005a) • 'high-level' comparison of generated and real corpora (Ai and Litman, 2006; Schatzmann et al, 2005a; Scheffler and Young, 2001). • HMM similarity of real and generated dialogues (Cuayáhuitl et al, 2005). • Cramér-von Mises divergence (Williams, 2007)	• policy performance in simulation (Ai et al, 2007b; Lemon and Liu, 2007; Schatzmann et al, 2005b)
Reward function:	• goodness-of-fit (Ai and Litman, 2007; Möller et al, 2007).	• prediction performance (Engelbrecht and Möller, 2007; Walker et al, 2000).
Whole sim. environment:		• transferability of policy performance when testing with real users (Lemon et al, 2006a)

Table 7.1 Utilised evaluation metrics for different simulated components used for RL-based strategy learning

In addition, we need to take into account that simulated environments are used in two different contexts: on the one hand they are used for strategy training, and on the other hand they are used for strategy testing. We argue that for strategy training it is more important that the simulated environments guarantees enough coverage, while for testing it is more important that the simulated environment is more realistic.

Furthermore, the evaluation should also account for the fact that the simulations are learned from WOZ data. We therefore introduce three separate evaluation phases to assure the quality of our simulated components:

1. We *directly* evaluate each single component when learning them form WOZ data (as described in the rest of this Chapter).
2. We *indirectly* evaluate the simulated learning environment (as a whole) when testing strategy performance (as described in Section 7.11 and Chapter 8).
3. We *post-evaluate* single components as well as the whole simulated environment by comparing dialogue generated in simulation with dialogues obtained with real users (as described in Chapter 8.5).

7.1.2 Outline

This Chapter describes a framework for training and testing a RL-based policy in simulation, where the simulated environment is obtained from limited amounts of WOZ data. We learn an optimised policy addressing the complex and challenging problem of *how many* database search results to present to the user, *when* to present them, and *which modality* to use for presentation, given the competing trade-offs between the number of results (large lists are difficult for users to process especially when only presented verbally), the length of the dialogue (long dialogues are tiring, but collecting more information can result in more precise results), and the noise in the speech recognition environment (in high noise conditions accurate information is difficult to obtain).[2]

As outlined in the framework in Chapter 5, we obtain a simulated learning environment from WOZ data, where all the simulated components are constructed in a data-driven manner. In particular, we create the action set by exploring the actions taken by the wizards, as described in Section 7.3. We use automatic feature selection techniques in order to define the state space, as described in Section 7.4. We then formulate the problem as an hierarchical Markov Decision Process (MDP), as explained in Section 7.5. In Section 7.6 we describe the baseline strategy obtained via Supervised Learning. In Section 7.7 we present a method for simulating channel noise if training data is sparse. In Section 7.8 we also introduce methods to build and evaluate user simulation from small data sets. The applied reward function is obtained using a modified version of the PARADISE framework, as described in Section 7.9. In Section 7.10 we explain how and why some parts of the state space need

[2] Note, that here we extend the problem addressed in Chapter 4, by addressing the question of which output modality to choose for generation.

to be quantised. In Section 7.11 we test and present the learned optimised policy, and we conclude in Section 7.12. We now describe the employed database.

7.2 Database Description

Here we learn a strategy for a specific application environment with one fixed database. (In contrast to Chapter 4 where we provided a general proof-of-concept by simulating various kinds of database retrieval.) The employed database is similar in ambiguity and structure to the one used in the WOZ experiment, but is smaller in size. Ideally, one would use the same database, however, the WOZ database contains over one million songs, which is (currently) unrealistic for a Spoken Dialogue System, as the lexicon for ASR and language modelling would exceed current system capabilities. Furthermore, the WOZ database contains many German names and titles, and the language of the implemented system is English.[3] However, the created database contains a high level of ambiguity, which is comparable to the ambiguity in the WOZ database. In other words, the returned number of matching items for an ambiguous user request stays about the same for both experiments.

The created database contains 3 genres, 10 artists, and 36 albums and 438 songs. The database contains ambiguities of two different types: semantic and referential. For learning we deal with the latter. With ‘referential’ ambiguity we mean ambiguity which is caused by under-specification of the user’s goal. For example, if the user asks for a ‘Blues’ song, there are 142 possible candidates. Referential ambiguity exists for every database feature (genre, artist, album, and song title). For example, there are 60 song names which are ambiguous between different artists or albums (6 of them being triply ambiguous and 1 song title has 4 instances), plus ambiguities which arise if the user only specifies partial information (or only a few words are recognised). For example, if the system only recognises the first two words of ‘Subterranean Homesick Alien’ by Radiohead, the song ‘Subterranean Homesick Blues’ by Bob Dylan is returned as well.

With ‘semantic’ ambiguity we refer to parsing ambiguities, where an entity can specify an album name or a song name (e.g. ‘Kid A’ is a song title as well as an album title by the artist Radiohead). Semantic ambiguities can also result from partial parses, for example, when a string is part of two different database categories, such as a song title referring to the artist (e.g. ‘Bob Dylan’s Blues’) or an album name referring to the artist (e.g. ‘The Freewheelin’ Bob Dylan’). In this work semantic ambiguities are resolved using hand-coded strategies. Every time more than one possible interpretation is returned from the parser the system engages in a clarification dialogue listing all the possible interpretations (e.g. “Do you mean album or song name?”).

[3] This switch of languages is simply because the WOZ tests are conducted at Saarland University in Germany, whereas the user tests were performed at Edinburgh University in the UK.

7.3 Action Set Selection

7.3.1 Method and Related Work

The action set defines the set of possible choices available to the learner at each state (see Chapter 3.2.1). Conventionally, the action space is predefined by the system the training data is taken from. Previous research either only optimises a sub-set of this predefined scope, e.g. (Singh et al, 2002; Walker, 2000), or optimises all actions available to the system, e.g. (Henderson et al, 2008).

In this work, however, we use WOZ data where the wizards talk freely, i.e. they are not constrained by a script but react intuitively. Hence, we are not limited to a predefined set of actions, but we are able to study human behaviour first, before we fix the system's functionality (a method which is also suggested by Williams and Young (2004b) and Levin and Passonneau (2006)). This approach also provides the advantage that it allows us to discover new action choices for system design. For error handling and clarification, for example, humans employ a much wider range of strategies than current systems do (Rieser and Moore, 2005). We therefore hope to gain insight about natural ways to seek and present information for dialogue systems from WOZ data. In particular, we are interested in the kind of information acquisition and multimodal presentation strategies human wizards (intuitively) apply.

We use manual annotation in order to discover the action set from WOZ data, following a method by Williams and Young (2004b). One challenge is here to find an adequate level of abstraction in order to describe the wizards' actions as a set of speech acts (SAs). On the one hand the SA taxonomy should cover the complexity of human behaviour. On the other hand, it should be simple enough to learn from a small data set.

In the following we propose a multilayered and multifunctional annotation scheme. This scheme is based on semantic annotations reflecting the discourse function of an utterance. The WOZ data is annotated with this scheme and a gold-standard annotation is obtained by measuring inter-annotator agreement. These low-level functions are then mapped / summarised into higher level dialogue acts, as described in the following.

7.3.2 Annotation Scheme

In order to classify the wizards' actions, we start out with a very fine-grained classification scheme which describes semantic discourse functions of so-called "Clarification Requests". The scheme was first introduced by Purver et al (2003a) and Rodriguez and Schlangen (2004) and extended by Rieser and Moore (2005). We found this scheme also to be suited for describing information acquisition and presentation strategies. We use a slightly modified version of the scheme, where we only use a subset of the suggested annotation tags, while adding another level describing the output modality, as summarised in Figure 7.2. In particular, the annotation scheme describes wizard actions in terms of their communication `level`, which describes the linguistic target after Clark (1996). We distinguish between utterances which aim to elicit `acoustic` information (e.g. *Sorry, can you please repeat?'* and utterances which aim to elicit further information to uniquely identify the user's `reference` (e.g. *By which artist?*). As well as utterances trying to establish `contact` (e.g. *Can you hear me?*), and utterances about the user's `intention` (e.g. *What do you want me to do?*). The problem `severity` describes which type of feedback the system requests from the user, i.e. asking for `confirmation`, for `repetition`, or for `elaboration`. The `modality` of the dialogue act can either be `verbal` or `multimodal`. Example 7.3.1 shows how a wizard action is annotated according to the scheme.

$$\left[\begin{array}{ll} \text{WIZARD SA:} & \left[\begin{array}{l} \texttt{level} \left\{\texttt{contact}|\texttt{acoustic}|\texttt{reference}|\texttt{intention}\right\} \\ \texttt{severity:} \left\{\texttt{confirmation}|\texttt{repetition}|\texttt{elaboration}\right\} \\ \texttt{modality:} \left\{\texttt{multimodal}|\texttt{verbal}\right\} \end{array}\right] \end{array}\right]$$

Fig. 7.2 Annotation scheme of discourse functions for wizard's actions

Example 7.3.1
User: *"Please search for titles by Madonna."*
Wizard: *"Please wait a moment [...] I found two-hundred and thirty seven items. The items are displayed on the screen"*
`level= goal, severity= confirm,`[4] `modality= multimodal`

An advantage of the applied scheme is that the described functions are strongly related to surface forms (i.e. a sequence of words), as shown by Purver et al (2003a); Rieser and Moore (2005); Rodriguez and Schlangen (2004). Thus, once the dialogue manager decides on the function, the corresponding surface generation can easily be determined.

Nevertheless, this detailed annotation scheme is likely to cause severe data sparsity, when learning a simulated environment from small data sets. For example, the learned user simulation has to be responsive to (at least) 24 system actions. We therefore summarise the detailed description of discourse functions into higher level dialogue acts based on the action combinations most frequently observed in the data, resulting in 6 distinct system actions, as defined in Table 7.2. (Also see Table 7.3 for examples from the corpus for each of the introduced SA categories.) Note that the chosen actions are a subset of the Speech Acts defined by the prominent DATE scheme (Walker and Passoneau, 2001).

[4] Note that asking the user to select an item on the screen is annotated as `confirm`.

The contact and intention level are excluded, since those levels are rarely observed in the data (also see next section).

Speech Act	Level	Severity	Modality
`reject`	acoustic	repetition	verbal
`explicitConfirm`	acoustic	confirmation	verbal
`askAQuestion`	goal	elaboration	verbal
`implicitConfirm`	goal	both	verbal
`presentVerbal`	goal	confirm	verbal
`presentMM`	goal	confirm	multimodal

Table 7.2 Reduced action set: System speech acts and corresponding discourse functions

7.3.3 Manual Annotation

We first evaluate the reliability of the low-level annotation scheme (Figure 7.2) via inter-annotator agreement.

A standard technique is to first produce an annotators' handbook, which contains detailed instructions for how to classify the utterances (Carletta et al, 1997). The corpus is then annotated (at least) twice, by an 'expert' and by a 'naïve' coder. The resulting annotations are then compared using the kappa (κ) statistics (Carletta, 1996). Annotations with $\kappa \geq .8$ are said to allow strong conclusions to be drawn, and annotations with $.8 > \kappa \geq .67$ allow tentative conclusions. These 'magical' thresholds are heavily discussed and criticised, e.g. (Craggs and McGee-Wood, 2005; Eugenio and Glass, 2004; Reidsma and Carletta, 2008).

In this work, we use the kappa statistic not only to evaluate the resulting annotation itself, but also to (indirectly) evaluate the reliability of the underlying coding scheme. In particular, we apply an iterative approach of annotation and re-annotation until the desired κ score is reached and a gold-standard annotation can be constructed. The reasoning behind this method is that linguistic categories are often arbitrarily assigned in a top-down manner. In order to discover the action set for learning, we want to create categories which have their foundation in the real data, i.e. they are created in a bottom-up approach.

In particular, we proceed as follows: Two annotators code all the instances in the data.[5] The cases where the annotators disagree are resolved through discussion. The coding scheme and the annotator's handbook are updated accordingly. A final version of the annotators handbook can be found in (Blaylock et al, 2006). The annotations reached a kappa score of $\underline{\kappa = .78}$ for the `level` dimension, and $\underline{\kappa = .71}$ for `severity` . The `modality` chosen by the wizard was automatically annotat-

[5] Many thanks to the second annotator Michael Wirth.

ed/logged as described in Section 6.3. The resulting annotations were then merged to a gold-standard annotation.

This gold-standard annotation was then summarised into 6 distinct Speech Acts (based on the action combinations most frequently observed in the data), as listed in Table 7.2 and corpus-based frequencies and examples are given in Table 7.3 (see discussion Section 7.3.2). 22.3% of the 793 wizard turns were annotated as presentation strategies, resulting in 177 instances for learning, where the six wizards contributed about equal proportions.

Note that this relatively small percentage of presentation strategies is mainly due to the following: First, the majority of the wizards' speech acts are related to providing task-related feedback, e.g. "OK, I'm searching", "OK, I've done that", "Just a moment please". We also did not annotate sub-dialogues considered with playlist building as here we are only interested in simple search tasks.

#	action type	frq.	%	example (original)	translation
1	`askAQuestion`	30	16.9	*Von welchem Künstler?*	By which artist?
2	`explConfirm`	5	2.8	*Du suchst das Lied 'Smile'?*	Are you searching for the song 'Smile'?
3	`implConfirm`	34	29.2	*Kannst du mir den Küstler des Stücks 'Summertime' nennen?*	Do you know the artist of the song 'Summertime'?
4a	`presentInfo` `-multimodal`	57	32.2	*Ich habe drei Alben. Ich zeige sie dir an.*	I have three (different) albums. I display them for you.
4b	`-verbal`	28	15.8	*Es gibt das Lied von 'Eric Clapton' oder von 'Eric Clapton & The Dominos'*	This song exists by 'Eric Clapton' or by 'Eric Clapton & The Dominos'.
⋆	`reject`	14	7.9	*Wie bitte?*	Pardon?
⋆	`intention`	9	5.1	*Was genau soll ich tun?*	What exactly do you want me to do?
		177	100%		

Table 7.3 System action types and frequencies as annotated in the data. (⋆) indicates the action types that won't be learned

7.3.4 Action Set for Learning

From the 7 high level acts we selected actions 1-4b for learning as these decisions involve complex trade-offs between different dialogue parameters (as further described below). The actions `reject` and actions related to `intention` recognition are excluded from learning. For the system action `reject` we decided to implement system behaviour which is based on a hand-coded heuristic, as this book does not specifically target error or uncertainty handling (see Chapter 2). Questions concerning intention recognition are also not addressed in this book . In general, the intention level is difficult to model, as discussed in (Ginzburg, 2008).

Actions 1-4b are organised to reflect the structure of information-seeking dialogues, as introduced in Section 3.4.1. The actions 1-3 belong to the *information acquisition* phase, where the wizards tried to gather new constraints from the user (with different emphasis on confirming the acquired information). The actions 4a and 4b belong to the *information presentation* phase and are different realisations for presenting a search result. Wizards present the results by either displaying a list while naming the number of search results verbally (`presentInfo-multimodal`), or they give a verbal summary of what was found (`presentInfo-verbal`). The case that the wizard only presented the items on the screen without saying anything does not occur in the WOZ data.

The learning problem is therefore defined as (a) *when* in the dialogue to present a list of items and *how many* items to present to the user and (b) *how* to present them (in which modality). In Section 7.5 we explain how to action set is formalised as a learning problem (using the Markov Decision Process framework).

7.4 State Space Selection

7.4.1 Method and Related Work

The state space defines the agent's view of the environment (see Chapter 3.2.1). Two main issues related to state space design are its scalability and that it is often selected manually (Paek, 2006).

The state space for learning has to be specified in advance by the dialogue designer. This is usually done manually. Choosing which state space features to include is non-trivial, as there is a trade-off between power of expression (many state space features can describe more complex problems) and keeping the learning problem tractable. This dilemma is also refereed to as the "curse of dimensionality" (see Chapter 3.2.2.3), which describes the fact that the number of possible states grows exponentially with the number of state variables (Young, 2002).

Previous research used different approaches to address this problem for state space design. Most of the previous work has used only a limited representation of the dialogue context, where the chosen features are mainly related to the task, for example a representation of slots that need to be filled and confirmed for this task, e.g. (Levin et al, 2000; Pietquin and Dutoit, 2006b; Singh et al, 2002; Williams and Young, 2007a).

In addition to these task-based state features dialogue designers have explored the use of enriched representations of dialogue context. One can distinguish between two approaches. One approach is to use a rich set of features, but reduce the possible state-action combinations by imposing some structure on the state space (cf. Chapter 3.2.2.3). (Henderson et al, 2005, 2008), for example, utilise the full dialogue context available and employ linear feature combination to group similar states together. Other research has investigated which individual context features help to learn better strategies, e.g. (Frampton and Lemon, 2006; Tetreault and Litman, 2006). How to select these features in the first place, however, is left to the designer's intuition and is still a manual task.

In the following we distinguish between quasi-standard task-based state features, and additional domain-specific state features. The former are a representation of the task as needed for dialogue management. Task-based features commonly represent the task-related slots, which are straightforward to enumerate manually, whereas selecting domain-specific features becomes the 'art' of state space design. It might take many iterations of training and testing (in simulation) until the 'right' feature combination (and their representation) is determined. In this book we investigate possible features for state space designs by first exploring the contextual features that human wizards condition their decisions on, and we use these features for state-space design.

7.4.2 Task-based State Space Features

We first manually enumerate a set of task-based variables which encode the dialogue task for learning. Task-oriented dialogue systems often use a simple semantic representation of the task as pairs of slots and fillers (also known as feature-value pairs). This representation is necessary for the dialogue manager to keep track of the task progress, i.e. how many slots have been filled, and their confirmation status. We choose a domain-independent representation of a 4-slot problem, corresponding to the task features *genre*, *artist*, *album*, *track* for the music-player domain (see Figure 7.3). In MDP-based strategy learning interpretation uncertainty is encoded in the confirmation (or "grounding") status of the slot values. POMDP-based strategy learning, in contrast, explicitly represents uncertainty in the state space representation, as described in Section 3.2.1.2. For MDP-based strategy learning it is standard practise to use slot-status categories such as "empty", "filled", "confirmed", because if all possible values of each slot would be considered as separate states, the number of states easily becomes unmanageable. We choose a representation where the `filledSlot` feature holds a binary value indicating whether a slot is filled. Equally, the state feature `confirmedSlot` holds a binary value indicating whether a slot is confirmed.

$$\left[\textit{task representation:} \begin{bmatrix} \texttt{filledSlot 1|2|3|4|:} \left\{ \texttt{0,1} \right\} \\ \texttt{confirmedSlot 1|2|3|4|:} \left\{ \texttt{0,1} \right\} \end{bmatrix} \right]$$

Fig. 7.3 State space selection: task representation

In general, there are two possible representations for this problem: Either to encode filling and grounding status into one variable, e.g. `slotN={empty|filled|`

`confirmed`}, or to encode this information into two separate variables, e.g. `filledSlotN`={`0|1`}, `confirmedSlotN`{`0|1`}. We choose the latter binary encoding approach, even though it results in a larger state space. However, binary encoding allows us to learn with linear function estimators (see Chapter 3.2.2.3). Linear estimators learn a preference scale over the attributes of one variable. That is, when encoding grounding information as {empty—filled—confirmed} one could either learn that confirmed is preferred to filled, and filled is preferred to empty, or the other way round. However, the preferred grounding status of a task variable highly depends on the current context, and therefore it can't simply be described by a linear ranking (as also illustrated in Chapter 4). Furthermore, the state space size for the binary representation is also limited by the fact that not all combinations of indicator truth values can occur in practise (e.g. a slot cannot be confirmed before it is filled).

7.4.3 Feature Selection Techniques for Domain-specific State Space Features

In Section 6.4 we formulated the hypothesis that good wizard multimodal strategies are context dependent. We now investigate whether wizards follow a specific context-dependent pattern, i.e. whether there are specific contextual features which influence the 'average' wizard's behaviour. We use these features to design the state space for RL.

In particular, we are interested in features which are related to *when* the wizards present the search results and *how* they present them. We therefore annotate the corpus with a rich set of dialogue context features using semi-automatic methods, as described in Section 6.3. We then use different feature selection techniques (see (Guyon and Elisseeff, 2003) for an introduction) to automatically select a subset of these features.

A similar approach to state space selection is also followed by Chickering and Paek (2007), who utilise Bayesian model selection techniques in order to choose state features from a large number of candidate variables. In contrast to the approach followed in this book, Chickering and Paek (2007) are not interested in exploring human behaviour, but they select features from a human-computer dialogue, which predict the (immediate) utility of an action.

7.4.3.1 Context/ Information-State Features

We first enumerate a rich set of dialogue context features to choose from.

A state or context in our system is a dialogue "information state" as defined in (Lemon et al, 2005). We divide the types of information represented in the dialogue information state into *local features* (comprising low-level and dialogue features), *dialogue history features*, and *user model features*. We also defined features reflecting the application environment (e.g. `driving`). The information state features are shown in Tables 7.4, 7.5, and 7.6, and are further described below. All features are automatically extracted from the XML log-files (as described in Section 6.3), and are available at runtime in ISU-based dialogue systems.

Local features:

First, we extracted features present in the "local" context of a wizard action, as shown in Table 7.4, such as the number of matches returned from the data base query (`DB`), whether any words were deleted by the corruption algorithm (see Chapter 6.2), the previous user speech act (`SA-type`) of the antecedent utterance, and its argument (`SA-argument`). Speech act type and action are manually annotated following a similar procedure as for annotating the system action set (see Section 7.3), with $\kappa = 0.68$ for `SA-type` and $\kappa = 0.76$ for `SA-argument`.

Note that the `deletion` feature (and later `delHist`, and `delUser`) counts the number of words deleted by the corruption tool (see section 6.1) and serves as an approximation to ASR confidence scores as observed by the system. Equally, the human wizard will be able to infer when words in a sentence were deleted and hence has a certain confidence that the input is complete.

Local features
`DB`: database matches (integer)
`deletion`: words deleted (yes/no)
`SA-type`: command, confirm, correct, request, provide
`SA-argument`: genre, artist, album, song, playlist, two arguments, three arguments, other, NULL

Table 7.4 Contextual/Information-state features: Local Features

Dialogue history features:

The history features account for events in the whole dialogue so far, i.e. all information gathered before entering the presentation phase, as shown in Table 7.5. We include features such as the number of questions that the wizard asked so far (`questHist`), how often the screen output was already used (`screenHist`), the average corruption rate so far (`delHist`), the dialogue length measured in turns (`dialogueLength`), the dialogue duration in seconds (`dialogueDuration`), and whether the user reacted to the screen output, either by verbally referencing (`refHist`), e.g. using expressions such as *"It's item number 4"*, or by clicking (`clickHist`).

Dialogue History Features
`questHist`: number of questions (integer)
`screenHist`: number screen outputs (integer)
`delHist`: average corruption rate; $\frac{no.wordsDeletedInDialogueSoFar}{no.utterancesInDialogueSoFar}$ (real)
`dialogueLength`: length in turns (integer)
`dialogueDuration`: time in sec (real)
`refHist`: number of verbal user references to screen output (integer)
`clickHist`: number of click events (integer)

Table 7.5 Contextual/Information-state features: History Features

User model features:

Under "user model features" we consider features reflecting the wizards' responsiveness to the behaviour and situation of the user. Each session comprises four dialogues with one wizard. The user model features average the user's behaviour in these dialogues so far, as shown in Table 7.6, such as how responsive the user is towards the screen output, i.e. how often this user clicks (`clickUser`) and how frequently s/he used verbal references so far (`refUser`); how often the wizard had already shown a screen output (`screenUser`) and how many questions were already asked (`questUser`); how much the user's speech was corrupted on average so far (`delUser`), i.e. an approximation of how well this user is recognised; and whether this user is currently driving or not (`driving`). This information was available to the wizards.

User model features
`clickUser`: average number of clicks (real)
`refUser`: average number of verbal references (real)
`delUser`: average corruption rate for that user; $\frac{no.wordsDeletedForUserSoFar}{no.utterancesForUserSoFar}$ (real)
`screenUser`: average number of screens shown to that user (real)
`questUser`: average number of questions asked to user (real)
`driving`: user driving (yes/no)

Table 7.6 Contextual/Information-state features: User Model Features

Note that all these features are generic over information-seeking dialogues where database results can be displayed on a screen; except for `driving` which only applies to hands-and-eyes-busy situations. Table 7.7 shows a context for the dialogue in Example 7.3.1, assuming that it was the first utterance by this user. This potential feature space comprises 16 features, many of them taking numerical attributes as values. Including them all in the state space for learning would make the RL problem unnecessarily complex. In the next two sections we describe automatic feature selection techniques, which help to reduce the feature space to a subset which is most predictive of *when* and *how* to present search results.

```
LOCAL FEATURES
        DBmatches: 1711
        deletion: 0
        SA-type: command
        SA-argument: artist
HISTORY FEATURES
        [questHist, screenHist, delHist,refHist,clickHist]=0
        duration= 10s
        length= 1 turn
USER MODEL FEATURES
        [clickUser,refUser,screenUser,questUser]=0
        driving= true
```

Table 7.7 Example: Features in the context after the first turn in example 7.3.1

7.4.3.2 Feature Selection

Feature selection describes the problem of selecting a subset of features that are most predictive for a given outcome. We use feature selection techniques to identify the context features which are most predictable for the wizards choosing a specific action. We choose to apply forward selection for all our experiments in order to not include redundant features, given our large feature set. We use 2 different types of selection techniques: We use the following feature *filtering* methods: correlation-based subset evaluation (CFS) (Hall, 2000) and a decision tree algorithm (rule-based SL). We also apply a correlation-based χ^2 *ranking* technique. Filtering techniques account for inner-feature relations, selecting subsets of predictive features at the expense of saying less about individual feature performance itself. Ranking techniques evaluate each feature individually. For our experiments we use implementations of selection techniques provided by the WEKA toolkit (Witten and Frank, 2005).

First, we investigated the wizards' information acquisition strategies, i.e. which features are related to the wizards' decision of *when* to present a list (`presentInfo`), i.e. the task is to predict `presentInfo` versus all other possible dialogue acts. None of the feature selection techniques was able to identify any predictive feature for this task. Hence, we conclude that there is no distinctive pattern the wizards follow for *when* to present information.

Next, we investigated the wizards' information presentation strategy, i.e. which features are related to the wizards' decision to present a list verbally (`presentInfo-verbal`) or multi-modally (`presentInfo-multimodal`). All the feature selection techniques consistently choose the feature `DB` (number of retrieved items from the database). Note that in Chapter 4 this information was already shown to be valuable for *when* to present the search results. Here the wizards primarily use this feature for *how* to present the results.

This result is maybe not very surprising, but it supports the claim that using feature selection on WOZ data delivers valid results. Relevant features for other domains may be less obvious. For example, (Levin and Passonneau, 2006) suggest the use of WOZ data in order to discover the state space for error recovery strategies. For this task, many other contextual features may come into play, as shown by (Gabsdil and Lemon, 2004; Lemon and Konstas, 2009) for automatic ASR re-ranking.

In sum, the dialogue state is now represented as follows. There are 8 binary state variables (for $1 \leq N \leq 4$): `filledSlotN` for whether each slot number N is filled, `confirmedSlotN` for whether each slot number N is confirmed, and one variable `DB` for the current number of DB hits, which takes integer values between 1 and 438, resulting in $\underline{2^8 \times 438 = 112,128}$ distinct dialogue states. In the next section, we combine the chosen state space (as described in this Section) with the chosen action space (as defined in Section 7.3) into a Markov Decision Process model in order to learn optimal, context-dependent action selection using RL.

7.5 MDP and Strategy Design

7.5.1 Motivation

A basic prerequisite to apply RL to dialogue strategy learning is to formulate the learning problem as a Markov Decision Process (MDP) model, or alternatively, as a Partially Observable Markov Decision Process (POMDP). This book uses MDPs for strategy presentation, as discussed in Section 3.2.1.2. Chapter 3.2.1 gives a brief overview on MDPs, introducing the main elements, namely the state space S, the action set A, the state transition function $\mathscr{T}$, as well as the reward function $\mathscr{R}$. In the following Section we present different techniques to impose additional structure on MDP models for dialogue strategy learning. Adding structure helps to reduce the possible state-action space and thus makes the learning problem more tractable (cf. Section 3.2.2.3). In particular, we investigate hierarchical RL, also known as sub-goal or sub-strategy learning (see (Barto and Mahadevan, 2003)) and learning with preconditions, implemented as Information State Update Rules (also see (Heeman, 2007)). These techniques can be used to encode prior knowledge which otherwise had to be learned in more complex ways.

7.5.2 Implementation

We use the REALL-DUDE development tool (Lemon et al, 2006c) for implementing the learning problem. REALL combines reactive planning and hierarchical RL (Shapiro, 2001). REALL is a language for defining reactive agent behaviour (i.e. reactive to a rapidly changing environment). It consists of a representation for expressing hierarchical, goal-oriented plans, together with an interpreter for evaluating those plans. REALL is also a learning system, containing different algorithms for simulation-based RL. Programmers can access this capability by writing plans with disjunctive elements, and embedding those choice points into hierarchical plans for learning. A plan is defined by the following elements. Each plan has an *objective*, which is the purpose of the plan, or the (sub-)strategy goal. It has *requirements*, which are the preconditions for action. For us, these preconditions are defined by Information State Update (ISU) rules. A plan can also have different *means* which are alternate methods for achieving the objective. For us, the different means correspond to the action set as defined in Section 7.3. Furthermore, the execution of a plan is based on a number of *features*, which correspond to the state space for learning (see Section 7.4).

The ISU rules are encoded by using the DUDE tool. DUDE is a "Dialogue and Understanding Development Environment" for rapid system development (Lemon and Liu, 2006). Section 8.1.1 provides a more detailed description of the tool. The ISU rules share the same state features with REALL. Thus, the implemented dialogue logic in DUDE directly triggers the preconditions for (sub-)plan execution in REALL.

7.5.3 Hierarchical Reinforcement Learning in the ISU Approach

$$
\begin{bmatrix}
\textit{acquisition action:} \begin{bmatrix} \texttt{askASlot} \\ \texttt{implConfAskASlot} \\ \texttt{explConf} \\ \texttt{presentInfo} \end{bmatrix} \textit{state:} \begin{bmatrix} \texttt{filledSlot 1|2|3|4|:} \left\{\texttt{0,1}\right\} \\ \texttt{confirmedSlot 1|2|3|4|:} \left\{\texttt{0,1}\right\} \\ \texttt{DB:} \left\{\texttt{1--438}\right\} \end{bmatrix} \\
\textit{presentation action:} \begin{bmatrix} \texttt{presentInfoVerbal} \\ \texttt{presentInfoMM} \end{bmatrix} \textit{state:} \begin{bmatrix} \texttt{DB low:} \left\{\texttt{0,1}\right\} \\ \texttt{DB med:} \left\{\texttt{0,1}\right\} \\ \texttt{DB high} \left\{\texttt{0,1}\right\} \end{bmatrix}
\end{bmatrix}
$$

Fig. 7.4 State-Action space for hierarchical RL

The structure of an information seeking dialogue system consists of an information acquisition phase, and an information presentation phase. For information acquisition the task of the dialogue manager is to gather "enough" search constraints from the user, and then, "at the right time", to start the information presentation phase, where the presentation task is to present "the right amount" of information in the right way– either on the screen or listing the items verbally. What "the right amount" actually means depends on the application, the dialogue context, and the preferences of users. For optimising dialogue strategies information acquisition and presentation are two closely interrelated problems and need to be optimised jointly: *when* to present information depends on the available options for *how* to present them, and vice versa.

We therefore formulate the problem as a MDP relating states to actions in a hierarchical manner (see Figure 9.4): 4 actions are available for the information acquisition phase; once the action `presentInfo` is chosen, the information presentation phase is entered, where 2 different actions for output realisation are available: presenting the list verbally (`presentInfoVerbal`) or multi-modally (`presentInfoMM`).

This structure follows the hierarchical dependencies identified action set (see Section 7.3.4). The available state features correspond to the feature set identified in Section 7.4. The state space for the information acquisition phase comprises the task-related state features (see Section 7.4.2), as well as a feature representing the number of retrieved database items (see Section 7.4.3). For information presentation, the `DB` feature is quantised into three distinctive ranges. The method for quantisation will be further described in Section 7.10. Note that for the acquisition phase the state-action space is still continuous. In total there are $\underline{4^{112,128}}$ (theoretically) possible policies for information acquisition. For the information presentation phase there are (only) $\underline{2^{2^3} = 256}$ possible policies. (Note, that the number of policies which are practically possible is much lower, due to different types of domain-specific and logical constraints, as further discussed in Section 7.11.4.)

The automaton in Figure 7.5 illustrates the action choices available to the learner at different times in the dialogue. The chosen system prompts for the final systems are modelled after the wizard utterance found in the corpus (see Table 7.3, also see example dialogues in Appendix A.3). Note that dialogue policy learning takes place at the intention level, i.e. system and user simulation are 'communicating' on the speech act level as shown in the example dialogues in Appendix A.2.

From a dialogue system perspective, the two sub-plans correspond to the Dialogue Management task of choosing an action ('what to say') and the Natural Language Generation (NLG) task of realising the action ('How to say it/How to choose the output'). For information seeking dialogues, Dialogue Management and NLG are two closely interrelated problems: the decision of *when* to present information depends on the available options for *how* to present them, and vice versa. We therefore optimise NLG and Dialogue Management policies in an integrated fashion.

Most of the previous work has learned strategies where the policy chooses between different utterances with one fixed surface form. In this book we de-couple form and function of a speech act, by learning in a hierarchical manner. First, the optimal function is determined, and then the optimal output medium. We currently only concentrate on whether to realise the presentation phase in an multimodal or verbal manner, while the surface realisation is still template-based.

7.5.4 Further System Behaviour

It is common practise to restrict the learning space of a policy by integrating prior domain knowledge in order to not have to learn the obvious (see for example (Heeman, 2007; Lemon et al, 2006a)), e.g. always `greet` the user at the start of the

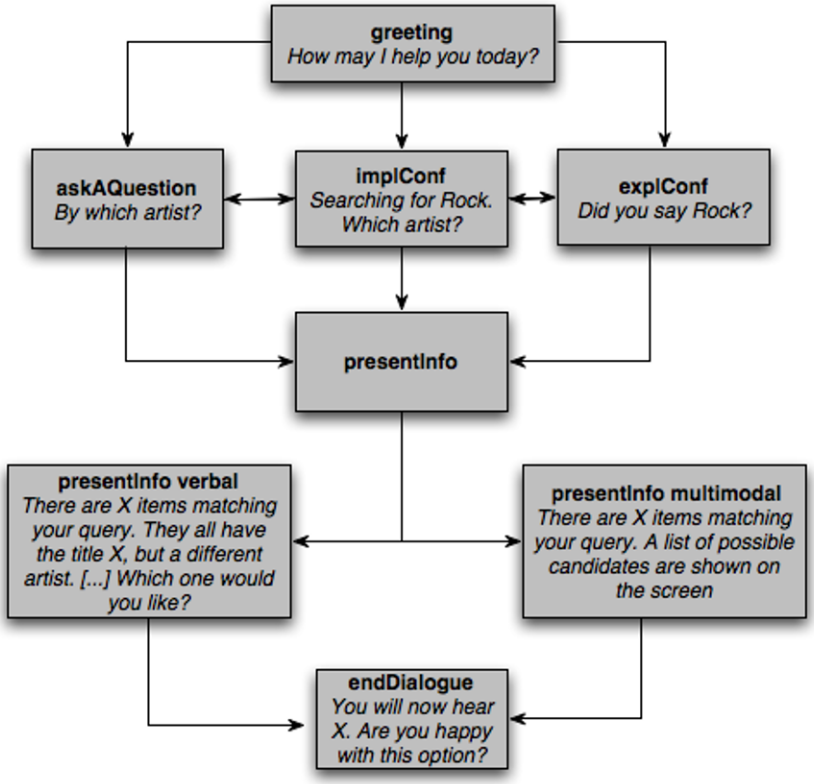

Fig. 7.5 Dialogue system automaton

dialogue. We implement this further system behaviour as ISU rules using the DUDE dialogue engine. The implemented rules encode simple restrictions on the available action choices, for example that only filled slots can be confirmed. An exhaustive list of preconditions for each action is shown in Table 7.8.

The task level is implemented assuming the following default ordering of slots: genre $\rightarrow$ artist $\rightarrow$ album $\rightarrow$ song. A similar ordering is also used by related work on in-car music player applications, e.g. (Forlines et al, 2005; Schulz and Donker, 2006; Varges et al, 2006; Wang et al, 2005).

The database is queried every time the user provides a new slot value, where all the filled slots serve as constraints for database retrieval. The retrieval also returns partial matches of strings, as discussed in Section 7.2.

Once the system decides to present the information, it automatically fills all empty slots, where a unique value can be retrieved. In example 7.5.1 the system identifies a unique artist and genre for the song, whereas the album still is ambiguous. Therefore, the system automatically fills genre and artist.

Action	Description	Precondition
`greeting`	Self-introduction and mixed-initiative opening (e.g. "*Hi this is* iTalk, *your talkative music player. How my I help you today?*")	Only available at the beginning of the dialogue.
`askASlot`	Ask for the next unfilled slot (default ordering) (e.g. "*Which genre would you like?*")	At least one slot still empty.
`implConf-AskASlot`	Implicit confirm the most recent filled slot while asking for the next empty slot (e.g. "*Searching for Rock music. By which artist?*")	The last provided slot value must be filled but not confirmed; at least one slot still empty.
`explConf`	Explicit confirm the most recent filled slot (e.g. "*Did you say 'Rock'?*")	The last provided slot value must be filled but not confirmed.
`presentInfo`	Present information (and enter the information acquisition phase) (choose between *multimodal* or *verbal* presentation)	At least one slot must be filled.
`endDialogue`	Serve the best matching item and end the dialogue. If the user rejects, the dialogue re-starts. (e.g. "*You will now hear 'Mozambique' by Bob Dylan. Are you happy with this option?* ")	User selected an item in the presentation phase.

Table 7.8 Available system actions and their preconditions

Example 7.5.1
User: *Search for the song "Layla" please.*
State: `filled [slot4], confirmed [], db:2`
System: *There are 2 results matching your query: all by the artist Eric Clapton, all from the genre Blues, all having the song title Layla, but a different album. You have the following options: Clapton Chronicles and MTV Unplugged. Which album would you like?*
State: `filled [slot1,slot2,slot4], confirmed [], db:2`

This basic system functionality exists independently from the learned strategy, i.e. the baseline strategy accesses the same basic functionality.

7.6 Wizard Behaviour

7.6.1 Method and Related Work

Our hypothesis is that simulation-based RL allows us to find optimal policies which are superior to those present in the original data. Therefore we create a policy which mimics the average wizard behaviour as a baseline. This allows us to measure the relative improvements over the training data (cf. Henderson et al (2008)).

A baseline system is a benchmark used as a basis for comparison. In order to make sensible claims, one must be aware that one compares dialogue policies which are commensurable (and not comparing “apples with oranges”). Furthermore, the observed differences in strategy should be informative, i.e. the differences should have a clear interpretation. In the following we discuss different options for a baseline strategy.

The most common practise is to compare the RL-based strategy against a hand-coded rule-based strategy, e.g. (Frampton and Lemon, 2006; Heeman, 2007; Lemon et al, 2006a; Scheffler and Young, 2002; Spitters et al, 2007; Walker, 2000). For hand-crafted strategies to be a fair basis for comparison they must be tuned to the same objective function, as pointed out by (Paek, 2006). That is, their parameters need to be manually optimised with respect to the same dialogue performance metric. We have followed this approach in Chapter 4, where we compared RL-based policies against various manually tuned hand-coded polices. Here, we choose an alternative baseline which allows us to measure *relative* improvements over strategies in the training data, as well as allowing us to compare human and optimised strategies.

(Paek, 2001) suggests comparing dialogue performance against WOZ data, where the wizard’s strategy serves as a “gold-standard”. Human performance is clearly superior for error recovery or NLU tasks, however, the wizards’ strategies cannot be regarded as being “gold standard” for information seeking tasks, as discussed in Chapter 6.4. Nevertheless, this comparison is still interesting as it allows to measure the difference between strategies intuitively applied by human wizards and strategies which are automatically optimised.

Another baseline is suggested by Singh et al (2002), which measures the relative improvement of the optimised model-based policy with respect to the initial exploratory strategies in the training data for the NJFun system. This comparison measures the *relative* performance gain when applying RL to automatic strategy optimisation. A similar approach is followed by Henderson et al (2008). They compare the learned policy against 8 expert systems, on whose data the policy was trained. In order to measure the relative improvement over the training data, (Henderson et al, 2008) had to ‘average’ over the different system strategies. This ‘average’ (or ‘multi-expert’) strategy is constructed using Supervised Learning, which models the majority decision the different systems take in a specific dialogue state (see Section 2.3.4).

Similarly, our data includes strategies of 5 different wizards, see Section 6.1. Thus, we first have to construct a baseline that reflects the ‘average’ wizard strategy in order to measure the relative the relative improvement over the initial data. We therefore follow the approach introduced by Henderson et al (2008) and apply Supervised Learning (SL), as described below.

Note that Supervised Learning is fundamentally different to Reinforcement Learning: RL is a statistical planning approach which allows us to find an optimal policy (sequences of actions) with respect to an overall goal (Sutton and Barto, 1998); SL in contrast is concerned with deducing a function from training data for predicting/classifying events. This book is not concerned with showing differences between SL and RL on a small amount of data, but we use SL methods to capture the average human wizard strategy in the original data, and show that simulation-based RL is able to find new policies, which were previously unseen.

7.6.2 Supervised Learning: Rule-based Classification

The following SL experiments are all conducted using the WEKA toolkit (Witten and Frank, 2005). As machine learning methods we choose a decision tree algorithm and a rule induction classifier since they are the easiest to interpret and also to implement in standard rule-based dialogue systems. Furthermore, both models are known to work well with relatively small data sets (Alpaydin, 2004). We learn with the decision tree J4.8 classifier, which is WEKA's implementation of the C4.5 system (Quinlan, 1993), and rule induction JRIP, which is the WEKA implementation of Cohen (1995)'s "Repeated Incremental Pruning to Produce Error Reduction" (RIPPER).

In particular, we want to learn models which predict the following wizard actions:

- Presentation timing: *when* the "average" wizard starts the presentation phase on a turn level (binary decision).
- Presentation modality: in *which modality* the list is presented (multimodal vs. verbal).

We use annotated dialogue context features as input, as described in Section 7.4.3.1 with feature selection techniques as described in Section 7.4.3.2. Both models are trained using 10-fold cross validation, comparing the predicted labels against the true labels in a hold-out test set. Table 7.9 presents the results for comparing the accuracy of the learned classifiers against the majority baseline.

A data analysis shows that all of the wizards are more likely to show a graphic on the screen when the number of database hits is ≥ 4. However, none of the wizards strictly follows that strategy.

	majority baseline	JRip	J48
presentation timing	52.00±2.20	50.23±9.70	53.49±11.72
presentation mode	51.00±7.04	93.53±11.47*	94.63±9.96 *
* statistically significant improvement, $p < .05$			

Table 7.9 Predicted accuracy for presentation timing and modality (with standard deviation ±)

Table 7.9 presents the results for comparing the accuracy of the learned classifiers against the majority baseline, constructed by always predicting the majority class in the data (which is a standard technique to compare SL classifiers). For presentation timing, none of the classifiers produces significantly improved results. Hence, we conclude that there is no distinctive pattern observable by the SL algorithms for *when* to present information. For strategy implementation we therefore use a frequency-based approach following the distribution in the WOZ data: in 48% of cases the baseline policy decides to present the retrieved items; for the rest of the time the system follows a hand-coded strategy.

For learning presentation modality, both classifiers significantly outperform the majority baseline. The learned models both learn the same rule set, which can be rewritten as in Algorithm 2. Note that this rather simple algorithm is meant to represent the average strategy as learned by SL from the initial data (which then allows us to measure the relative improvements of the RL-based strategy).

Algorithm 2 *SupervisedStrategy*

Require: An integer $DB \geq 1$.
1: DB $\Leftarrow$ number of retrieved database items
2: **if** $DB \leq 3$ **then**
3:     **return** `presentInfoVerbal`
4: **else**
5:     **return** `presentInfoMM`
6: **end if**

The learned rules are used to implement a rule-based baseline strategy (SL policy) by controlling the choice points of the implemented automaton (see last Section Figure 7.5). As noted above, the SL algorithms could not identify a common pattern in the data for when to start the presentation phase. We therefore employ a frequency-based approach following the distribution in the WOZ data: in 0.48 of cases the baseline policy decides to present the retrieved items; for the rest of the time the system randomly chooses between the 4 available actions for information acquisition (where the implemented system constraints assure that the strategy is still sensible). An example for the SL baseline policy (deployed in the simulated environment) can be found in Appendix A.2, Table A.2.

7.7 Noise Simulation: Modelling the Effects of Mis-Communication

7.7.1 Method and Related Work

One of the fundamental characteristics of Human-Computer Interaction is an error prone communication channel, as discussed in Chapter 2.1. Therefore, the simulation of channel noise is an important aspect of the simulated learning environment. In spoken dialogue systems, the channel noise, as introduced by Automatic Speech Recognition (ASR), is often measured in terms of word error rate (WER). WER is defined as in Equation 7.1, where S is the number of substitutions, D is the number of the deletions, I is the number of the insertions, N is the number of words in the reference.

$$WER = \frac{S+D+I}{N} \tag{7.1}$$

Previous work on dialogue strategy learning either ignores the error channel altogether, e.g. (Levin et al, 2000; Schatzmann et al, 2007c), or simulates errors using (statistical) models of different complexity. One can distinguish between two strands of research: methods that model noise at a globally fixed error rate, and methods which compute the word error rate for every utterance. The former assumes a fixed distribution of errors for the whole interaction. Pietquin and Renals (2002), for example, condition the error rate on the type of the task (e.g. word versus digit recognition). Prommer et al (2006) set a fixed error rate per individual speaker. The simulated word error rate can also be set to approximate the distribution found in the training data: In (Georgila et al, 2005a; Lemon et al, 2006a), for example, 70% of all utterances are transmitted via a WER of 0%, 10% of with a WER of 100%, and 20% with a varying rate between 0 and 100%. In (Scheffler and Young, 2001), the user's dialogue behaviour is modelled as a network of interconnected states, with user actions corresponding to state transitions and errors corresponding to special "mumble" or "null" transitions, which occur with a certain (globally fixed) frequency.

The other strand of research explicitly models the confusability of individual words and utterances to overcome the assumption of a globally fixed error rate. Error simulation based on phone-level confusions is explored by in different work, including (Deng et al, 2003; Pietquin, 2004; Stuttle et al, 2004). While experiments with phone-level confusions produce promising results, the amount of training data needed to model context-dependent phone confusions for a typical tri-phone based recogniser is often very large. Computationally less expensive word-level confusion methods have been suggested by Pietquin and Dutoit (2006b) and Schatzmann et al (2007b).

In this book we model channel noise on a level of abstraction which is adequate to the task and the available data: The introduced model is complex enough to learn a presentation strategy which is sensitive to the noise level, while it is obtainable from a small data set. This book does not focus on learning a (sophisticated) error handling strategy, but on learn an information seeking strategy in the presence of noise. In particular, we require a framework for modelling the effects of misunderstandings (as introduced by insertions and substitutions) and non-understanding (as introduced by deletions) on formulating a database query. Not all the the approaches listed above do distinguish between substitution, deletion, and instertion, e.g. (Pietquin and Dutoit, 2006b; Prommer et al, 2006) only model word confusions/substitutions.

Furthermore, we consider it to be desirable if the noise simulation model is language independent. The language of the WOZ data is German, while the user tests are conducted in English. It is therefore problematic to directly train a detailed noise simulation (e.g. based on word-level confusion) on the recorded WOZ interaction. In the following we present a task-based error model which simulates the general presence of noise and its effects on the interaction, rather than simulating the noise itself.

7.7.2 Simulating the Effects of Non- and Mis-Understandings

Here we describe a framework for simulating how non- and misunderstandings effect the behaviour of the user, as well as the system and the task. We first estimate a (global) error distribution from the data, following Georgila et al (2005a); Lemon et al (2006a). In the WOZ experiment 20% of the user utterances were corrupted by low noise and 20% by high noise. This is averaged into a 30% chance of the input being mis-recognised. In the WOZ setting only 7% of the user utterances lead to a (noticeable) communication error (see Section 6.2.3). However, the performance of present speech and language processing technology cannot be compared to human performance, especially not when communicating in noisy environments. A 30% error rate is a realistic assumption for deployed dialogue systems. For example, Carpenter et al (2001) find that, in the COMMUNICATOR corpus, 33% of over 4.5k user utterances had some kind of processing error. This is consistent with the findings by Georgila et al (2005a) who estimated a 30% error rate on 697 dialogues from the same corpus. Hirschberg et al (2001) report on similar results for the TOOT corpus by Litman and Pan (1999): 29% of over 2.3k user utterances were corrections of system turns.

For non-understandings we estimate a rejection rate of about 4% of the utterances. We model non-understandings by assigning a 4% probability of generating out-of-vocabulary (OOV) utterances to the user model (similar to Scheffler and Young (2001)'s "mumble" state transition). OOV utterances lead to an empty parse. Corpus studies report similar OOV rates (Bazzi and Glass, 2000).

While the simulation of non-understandings is pretty straight forward, the simulation of misunderstandings requires a more complex model. Where non-understandings are recognised immediately by the system (as well as by the user), misunderstandings might not be detected until a later turn or might not be detected at all. Results from a corpus study by Bohus and Rudnicky (2005a) for example show that there is a discrepancy of up to 51.5% between what is mis-recognised by the system and what is corrected by the users: users may correct right hypothesis and may leave errors uncorrected. In order to model such discrepancies, we separately model how misunderstandings effect the user and the system side.

For the user side, the noise model sets the probabilities of the user rejecting or accepting the system's hypothesis for the user simulation. In general, the probabilities for the user simulation are learned from data (as described in the next Section 7.8). Note that the user speech act labels in the WOZ data are held on an abstract level, i.e. they still need to be realised in a context-specific way, e.g. with respect to the current noise level. In particular, the two abstract user speech acts `repeat content` and `yes-no answer` are differently realised depending on the noise model, as shown in Table 7.10. For example, if the user simulation predicts a user's speech act as a yes-no answer, the noise model sets the probability for the user agreeing with 'yes', or rejecting with 'no'. That is, while the noise model follows a fixed error rate, the user simulation determines the probability of individual speech acts in a data-driven context-dependent manner.

Annotated in corpus	No error	Semantic error
`repeat content`	confirm and re-provide same	negate and re-provide different
`y/n answer`	yes-answer	no-answer

Table 7.10 Action mapping from corpus annotation to user acts including the noise model

For the system side, the effects of misunderstandings are modelled by how we estimate task completion in the reward function. We estimate a 30% chance for each slot which is filled but not confirmed for being incorrect due to mis-recognition. We account for this fact when calculating Task Success in the reward function, as described in Section 7.9. For slots which are filled but confirmed we always assume that those are correct. In future work we will refine this assumption.

In sum, we use a simple model simulating the effects of non- and misunderstanding on the interaction, rather than the noise itself. We simulate the *effects* that noise has on the user behaviour, as well as for the task accuracy. This method is especially suited to learning from small data sets. Furthermore, the model is easily adjustable for different levels of noise. Lemon and Liu (2007), for example, use a similar implementation for the noise model in order to investigate the effects of training and testing a strategy in environments with different noise levels. The drawback of the applied technique is that noise is modelled at a globally fixed error rate. It does not adjust the ASR quality according to the local (e.g. phonetic) context. One direction for future research is to simulate a context-dependent semantic error rate following work by Bohus and Rudnicky (2002), which predicts semantic confidence levels, rather than simulating WER.

7.8 User Simulation

A "user simulation" is a predictive model of real user behaviour used for automatic dialogue strategy development. The major advantage of user simulations is that they are less cost-intensive than experimenting with real users.

It should be noted that *user simulation* is different to *user modelling*, although these two terms are sometimes confused in the literature. The main difference is that they serve different functions on the system side. The purpose of user modelling is to adapt strategies to different user *types*, or to compute state updates in POMDP systems. For example, dialogue strategies are adapted to user expertise (Hassel and Hagen, 2005), or information is presented according to user preferences (Demberg and Moore, 2006). Note that RL can also be used for user modelling in order to learn user-type-specific strategies. For example, different user profiles can be incorporated in the reward function, as demonstrated in Chapter 4.2.5.

Recent work on user simulation for strategy learning differs from user type in various respects. The role of a user simulation is as a synthetic dialogue partner which generates behaviour representative for a diverse user population. As such, the user simulation must be able to predict user responses that real users might give in the same or a similar context. In addition, it must also represent the statistical distribution over all possible types of user behaviour in order to account for natural variety in real human user behaviour. These requirements clearly favour a statistical approach, where the model's parameters can be estimated from real data.

This Section proceeds as follows. In Subsection 7.8.1 first review user simulations in the context of dialogue strategy learning. We then explain how user acts are annotated in the WOZ data in Subsection 7.8.2. In Subsection 7.8.3 we then show that building a simple bi-gram model is not a good option, as it leads to severe data sparsity. We therefore introduce two different methods for building user simulations for learning from small data sets: in Subsection 7.8.3 we present a novel technique for cluster-based user simulations; in Subsection 7.8.5 we apply smoothing to the bi-gram model. We then evaluate the user simulations using the Kullback-Leibler (KL) divergence. In Subsection 7.8.7 we describe how the user goal is released with respect to the chosen user action.

7.8.1 Method and Related Work

7.8.1.1 User Simulation for Dialogue Strategy Learning

Statistical user simulation for dialogue strategy learning is explored by a number of research groups, not only in the context of RL, but also in combination with supervised techniques for strategy optimisation, e.g. (Filisko and Seneff, 2005, 2006), or in order to identify weak spots during (manual) strategy development (Chung, 2004a), or also for automatic dialogue evaluation (Engelbrecht et al, 2009; López-Cózar et al, 2003; Möller et al, 2006; Watanabe et al, 1998). Also see (Schatzmann et al, 2006) for a review of user simulation for dialogue strategy learning.

For simulation-based (also known as model-free) RL, the dialogue system learns online during the interaction with the simulated user. One of the major advantages of simulation-based RL is that a vast state-action space can be explored, as discussed in Chapter 3.2.3. A simulated user allows any number of training episodes to be generated, so that the learning dialogue manager can exhaustively explore the space of possible strategies.

In general, one can distinguish user simulations according to the employed level of abstraction, their intended purpose, and the utilised modelling technique. The level of simulated interaction can either be the acoustic level, e.g. (Chung, 2004a; Filisko and Seneff, 2005, 2006; López-Cózar et al, 2003), the word-level, e.g. (Araki et al, 1997; Watanabe et al, 1998), or the intention level, as first introduced by (Eckert et al, 1997). Simulations on the word and acoustic level are mainly used for dialogue system testing (including the ASR and NLU module). Simulations on the intention level are most popular for RL-based strategy learning, as they outperform the lower level approaches in terms of robustness, portability and scalability. A variety of different statistical techniques are applied to build such intention-based simulations. All of them exploit the Markov assumption, i.e. the next user action is assumed to be independent from the dialogue history and can be predicted based on some representation of the current dialogue state. This is formalised as a sequence of state transitions and dialogue acts: at any time t, the user is in a state S, takes action a_u, transitions into the intermediate state S, receives system action a_s, and transitions into the next state S' where the cycle restarts.

$$S \to a_u \to a_s \to S' \to \dots \tag{7.2}$$

(Eckert et al, 1997, 1998) pioneered the research area by using simple bi-gram models to predict the next user action based on the previous dialogue state. Much current work followed, e.g. (Scheffler and Young, 2001, 2002), (Georgila et al, 2005a, 2006b), (Cuayáhuitl et al, 2005), (Pietquin, 2004; Pietquin and Dutoit, 2006a,b), (Schatzmann et al, 2005a,b, 2007a,c), (Ai and Litman, 2006, 2007; Ai et al, 2007b), (Rieser and Lemon, 2006a) and so on.

Many of these methods define a "user goal" in order to ensure that the simulation model generates realistic behaviour in a consistent and goal-directed manner. In this context, "user goal" usually refers to a concrete task the user tries to accomplish by using the system. It is commonly assumed that the user goal is an instance of the task space defined by the possible slot values.

One major issue for user simulation is to assure "consistent" behaviour. This can mean consistency with respect to the current user goal. For example, Schatzmann et al (2007a,c) introduce agenda-based models where the user goal is modelled as a hidden variable to assure consistency. For other domains, such as tutoring, Ai and Litman (2007) introduced knowledge-consistent user simulations.

Consistent user behaviour also means being concordant with the dialogue history. Due to the Markov assumption the user simulation is "memory-less" towards its own previous behaviour. For example, a user simulation might first confirm a certain value and then negate it later. (Note that real user behaviour can also be inconsistent to a certain degree, as shown by (Bohus and Rudnicky, 2005a)). Previous work has addressed this issue by modelling longer sequences of interactions. For example, Scheffler and Young (2002) propose the use of graph-models in order to improve temporal consistency. Georgila et al (2005a, 2006b) combine higher level n-gram models with linear feature combination techniques, and thus are able to learn with a rich representation of dialogue context and dialogue history. Work by Cuayáhuitl et al (2005) uses Hidden Markov Models to simulate user behaviour as a sequence of actions. Pietquin (2004); Pietquin and Dutoit (2006a,b) use Bayesian Networks to allow multiple dependencies between dialogue states and user actions. However, the drawback to all these methods is that they need to be trained on large amounts of data.

7.8.1.2 Learning User Simulations from Small Data Sets

In this work we are challenged by learning user simulations from a small data set. Previous work has also addressed the question of how to learn user simulations when training data is sparse. Prommer et al (2006) for example learn a simple bi-gram model from a data set comprising 82 dialogues and 314 utterances. The bi-gram model comprises 4 user actions and 8 system actions, where 53.13% of the possible combinations have zero frequency. As a consequence the user simulation will never generate the full range of user actions (after a specific system action), and thus fails to let the learner sufficiently explore the state-action space (also see discussion Section 7.1.1).

An approach to overcome the data-sparsity problem is suggested by (Schatzmann et al, 2007c). They use a summary-space mapping approach to learn a user simulation from 160 dialogues and 6452 turns. In the summary space mapping approach similar states are summarised (using the Expectation-Maximisation algorithm), and a distribution of possible user behaviour is assigned to a set of states. This method allows generation of the full range of possible user behaviour in every state. In (Rieser and Lemon, 2006a) we introduced similar approach to Schatzmann et al (2007c), which is described in the following.

7.8.1.3 Training and Testing with Simulated Users

A common strategy is to first get promising results in simulation, before testing with real users, e.g. (Filisko and Seneff, 2006; Frampton, 2008; Lemon et al, 2006a). However, it was criticised that training and testing on the same user simulation is "cheating" (Paek, 2006). As pointed out by Schatzmann et al (2005b), policies trained with a poor user simulation model may appear to perform well when tested on the same model, but fail when tested on a better user simulation model. Fortunately, the converse is not true: policies learned with a complex model will still perform well when tested on a less complex model. Lemon and Liu (2007) confirm that policies for "worst case scenarios" (e.g. an uncooperative user, or noisy conditions) transfer well to "easier" situations, but not the other way round. In sum, these results indicate that in general training and testing should be done with two separate user simulations which meet different requirements.

Evaluating simulated components is a difficult issue, as already discussed in Section 7.1.1. On the one hand the evaluation metric needs to assess the coverage provided by the simulation (in order to learn robust strategies); on the other hand the metric needs to assess how realistic the simulation is in order to produce realistic feedback. These issues become even more prominent when learning from small data sets. Furthermore, the importance of coverage and realism seem to be differently weighted for strategy testing and strategy training.

Results by Ai et al (2007b) indicate that for strategy training strategy coverage is even more important that realistic feedback. (Ai et al (2007b) show that random models perform better than more accurate ones if the latter fail to provide enough coverage.)

The main purpose of user simulation used for strategy testing is to get a realistic estimate of strategy performance when tested with real users, see for example (Lemon et al, 2006a). Hence, user simulation for strategy testing needs to be more realistic.

7.8.2 User Actions

In order to learn user actions from data, they first have to be annotated. Thus, an action set for annotation must be defined, similar to the system action set described in Section 7.3. Similarly, the user model's action set must achieve a trade-off between adequately covering the complexity of human behaviour and being simple enough for its parameters to be trainable on a limited amount of corpus data (in analogy to the system action set). We therefore employ the same iterative manual annotation method as for the system actions (see Section 7.3.3). The annotations reach a kappa score of $\kappa = .70$.

Table 7.11 User action types and frequencies as annotated in the data

#	action type	freq	%	example (original)	translation
1	`add`	54	30.5	*ah, Ella Fitzgerald.*	*erm, Ella Fitzgerald.*
3	`repeat`	57	32.2	*ja, Smile ja.*	*yes, Smile yes.*
2	`y/n`	14	7.9	*ja, in Ordnung.*	*yes, that's OK.*
4	`change`	17	9.6	*dann machen wir was anderes und zwar hätte ich gern eine Playlist mit drei Liedern.*	*Let's try something else then. I would like a playlist with three songs.*
	`others`	35	19.8	—	no answer, comment, aside

For our domain, the user can either add new information (`add`), repeat or paraphrase information which was already provided at an earlier stage (`repeat`), give a simple yes-no answer (`y/n`), or change to a different topic by providing a different slot value than the one asked for (`change`). Examples from the corpus are given in Table 7.8.2 and in the dialogues listed in Appendix A.1.

The majority of user speech acts are to add or to repeat information. Topic change and yes-no answers are less common. Instead of providing a simple yes-no answer the users often repeat or paraphrase the information. Note that the speech acts subsumed under `others` are not (only) out-of-domain utterances (like giving no answer, making a comment aside), but also any user action which is handled by hand-coded strategies in the final system, e.g. the user asking for help, repeat, or quit (see Section 8.1.1). The users also had the option to select an item by clicking on the screen. As clicking only is observed 11 times over the whole WOZ experiment (as discussed in Chapter 6.4), user clicks are subsumed under the action `add` (adding information).

7.8.3 A Simple Bi-gram Model

We first construct a simple bi-gram model in order to explore quality of the data. Bi-gram (or more general n-gram) models for user simulations were first introduced by (Eckert et al, 1997, 1998). An n-gram based user simulation predicts the user action $\widehat{a}_{u,t}$ at time t that is most probable given the dialogue history of system and user actions (see Equation 7.3 where $a_{s,t}$ denotes the system action at time t). In practise, data sparsity may prohibit the use of long dialogue histories. Therefore, the full history can be approximated using a bi-gram model by using the Markov assumption, as in Equation 7.4.

$$\begin{aligned} \widehat{a}_{u,t} &= argmax_{a_u,t} P(a_{u,t}|a_{s,t-1}, a_{u,t-1}, ..., a_{u,0}, a_{s,0}) && (7.3) \\ &\approx argmax_{a_u,t} P(a_{u,t}|a_{s,t-1}) && (7.4) \end{aligned}$$

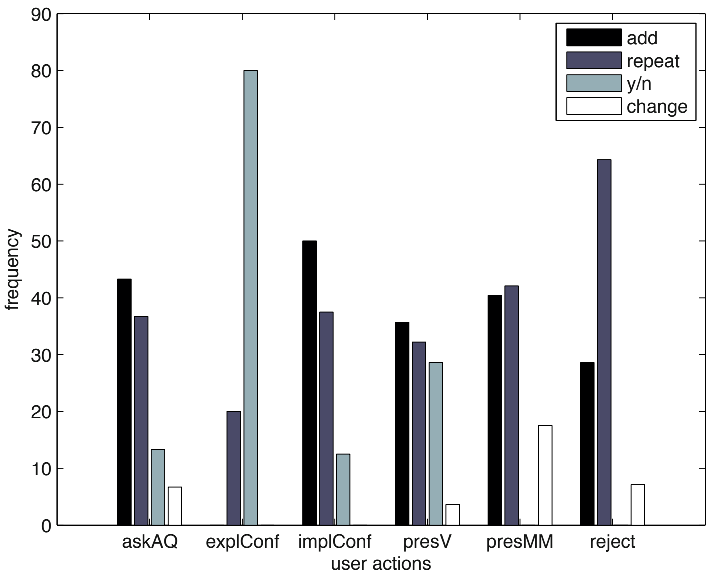

Fig. 7.6 User action frequencies following a system act (bi-gram model): 25% zero frequencies for state-action pairs due to data sparsity

Although the bi-gram model only relates one state/feature to one action, it still can result in data sparsity for small data sets. The bi-gram model obtained from our WOZ data and the observed frequencies are shown in Figure 7.6. When examining the distributions of user replies per system turn for the bi-gram model, we can see that in 25% of the state-action pairs have zero frequencies. For comparison, the bi-gram model by Prommer et al (2006) has zero frequencies for 53.13% of the state-action pairs in their data set.

As discussed before, user simulations for automatic strategy learning need to cover the whole variety of possible user actions for each state in order to produce robust strategies. In other words, user simulations should allow the learner to also find strategies which are not in the data. Especially when learning from small data sets, user simulations for automatic strategy *training* should cover the whole variety of possible user actions for each state in order to produce robust strategies. Ai et al (2007b), for example, show that random models outperform more accurate ones if the latter fail to provide enough coverage. On the other hand, user simulations used for *testing* should be more accurate with respect to the data in order to test under realistic conditions, e.g. (Möller et al, 2006).

We therefore apply two learning methods to deal with data sparsity (for n-gram models): First, we develop a user simulation which is based on a new clustering technique; second, we apply smoothing (which is the standard technique applied to account for zero frequencies in n-gram models).

7.8.4 Cluster-based User Simulation

We introduced a cluster-based technique for building user simulations from small amounts of data in (Rieser and Lemon, 2006a). A similar approach has later been suggested by Schatzmann et al (2007c), called the "summary-space mapping technique", where similar states are summarised, and a distribution of possible user behaviour is assigned to a set of states, which we call "clusters". This method allows one to generate the full range of possible user behaviour in every state.

Cluster-based user simulations generate explorative user behaviour which is similar but not identical to user behaviour observed in the original data. In contrast to the bi-gram model, where the likelihood of the next user act is conditioned on the previous system action, the likelihood for the cluster-based model is conditioned on a cluster of similar system states, see Equation 7.5.

$$\widehat{a}_{u,t} \approx argmax_{a_u,t} P(a_{u,t} | cluster_{s,t-1}) \tag{7.5}$$

The underlying idea is that, with sparse training data, we want user simulations to be "similar to real users in similar situations" but not identical, since we do not want to over-fit the data. (Note that for n-gram models the trade-off between coverage and accuracy/data fit is also known as the bias-variance trade-off (Alpaydin, 2004).) This user simulation should generate any kind of observed user behaviour in a context (as opposed to the zero frequencies in the bi-gram model), while still generating behaviour which is realistic/ pragmatically plausible in this situation. That is, we want our user simulation to generate behaviour which is *complete* and *consistent* with respect to the observed actions. We also want our model to generate actions which show some *variability* with respect to the observed behaviour, i.e. a controlled degree of randomness. This variance will help us to explore situations which are not observed in the data, which is especially valuable when building a model from sparse training data, cf. (Ai et al, 2007b).

Clustering is applied in order to identify more general situations than the previously annotated system speech acts by grouping them according to their similarity. Cluster analysis is a un-supervised machine learning technique (see definition Section 2.3.1) which partitions the data into subsets (clusters). In contrast to supervised learning there are no predefined class labels, but the labels are constructed in a bottom-up manner by grouping feature vectors based on their similarity. For building such clusters we apply the Expectation-Maximisation (EM) algorithm. The EM algorithm is an incremental approach to clustering (Dempster et al, 1977), which fits parameters of Gaussian density distributions to the data.

The first step, calculation of the cluster probabilities (which are "expected" class values) is "expectation"; the second, calculation of the distribution parameters, is "maximisation" of the likelihood of the (Gaussian) distributions given the data.

7.8.4.1 Semantic Annotation of System Acts

In order to define similarity between system actions, we need to describe their (semantic) properties. In Section 7.3 we explained that for the incremental annotation of system acts we start out with a fine grained scheme of "Clarification Requests" (Rieser and Moore, 2005; Rodriguez and Schlangen, 2004), which is then summarised to a smaller set of more general system acts (cf. Figure 7.2, Section 7.3). We now reverse this process and decompose the system actions into a more detailed semantic description.[6]

7.8.4.2 Results

The EM algorithm generates three state clusters, as shown in Figure 7.7. The system acts `askAQuestion` and `implConfirm` are summarised into cluster 1; `explConf` and `reject` are in cluster 2; and `presentListVerbal` and `presentListMM` are in cluster 3.

For every cluster we assign the observed frequencies of user actions (i.e. all the user actions which occur with one of the states belonging to that cluster), as shown in Figure 7.8. We observe the following differences when comparing the cluster-based frequencies with the ones from the original bi-gram model (see Figure 7.6): First,

[6] Note that such a flexible, tree-based approach to utilise multifunctional and multidimensional dialogue act annotation schemes is also suggested by Bunt (2007).

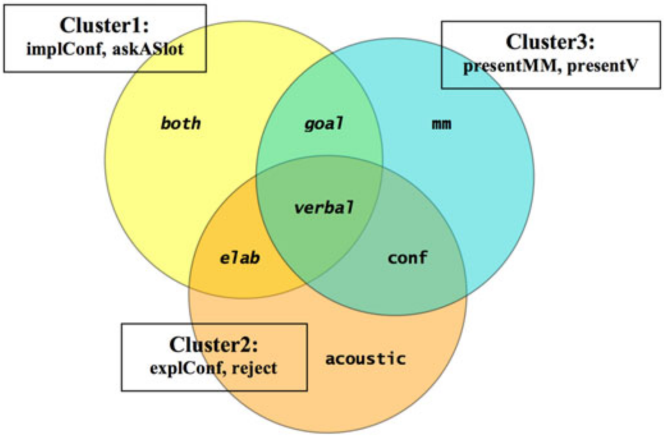

Fig. 7.7 Clustered System Actions

there are no zero frequencies for any of the possible user-system act combinations. The cluster-based user simulation generates the full range of possible user acts in any state, and thus allows to learn robust strategies. Second, the previously 6 states are now summarised into 3 states. Consequently, the cluster-based user simulation will generate the same behaviour for the system acts being in one cluster.[7]

7.8.5 Smoothed Bi-gram User Simulation

For our second user simulation model we apply smoothing to a bi-gram model. Smoothing is a standard technique applied to n-gram models to account for zero frequencies observed in the data. Smoothing assigns a non-zero probability to zero-probability and low-probability n-grams. We implement a simple smoothing technique called "add-one smoothing" (Jurafsky and Martin, 2000). This technique discounts some non-zero counts in order to obtain probability mass that will be assigned to the zero counts. We apply this technique to the original frequency-based bi-gram model. The resulting model is shown in Figure 7.9. In comparison to the

[7] Note that for each two system acts the user simulation now reacts with the same likelihood. Nevertheless, the learner needs to experience the different consequences/feedback on his actions, in order to learn distinctive behaviour. These differences still do exist by way of the different effects that a user action has on the state space (dependent on the previous system act). These effects are implemented by logical constraints and update rules for system behaviour, as presented in Section 7.5.3. For example, if the user adds new information after `implConfirm` a slot gets filled and another gets confirmed in the system state space, whereas after `askAQuestion` a slot only gets filled. These different sate space representations have different expected rewards, thus the strategy learns distinctive behaviour.

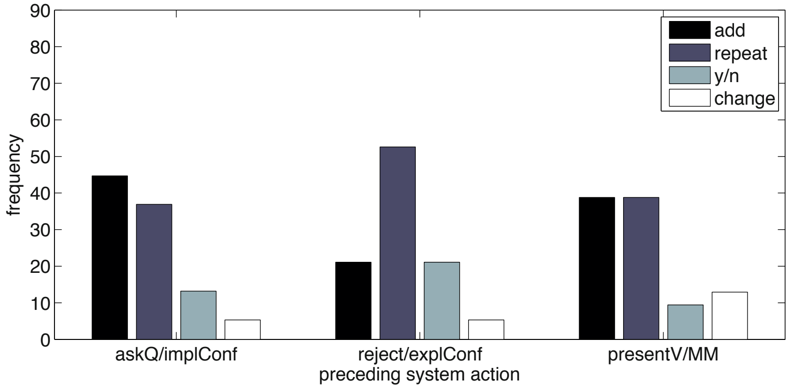

Fig. 7.8 User action frequencies from the cluster-based user simulation

original bi-gram model the smoothed model has no zero frequencies. In contrast to the cluster-based model, the smoothed model still has 6 distinct states, but for some state-action combinations the predicted frequencies are still low. We therefore expect this model to produce less exploratory behaviour. Furthermore, the frequency for the non-zero states are modelled from data from a relatively small user population. Thus, this model is more likely to overfit.

In general, the smoothed model seems to be closer to the original data than the cluster-based one (thus being more realistic at the expense of allowing less exploratory behaviour). In the next section we introduce an evaluation metric which allows us to assess the level of exploratory versus realistic user behaviour as predicted by the different user simulations.

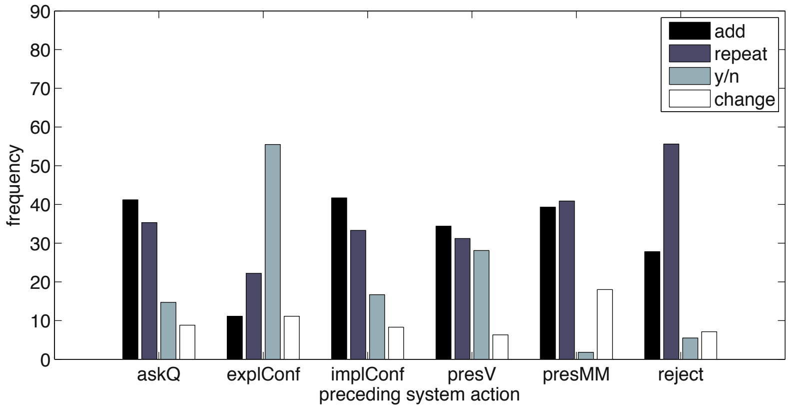

Fig. 7.9 User action frequencies from the smoothed bi-gram user simulation

7.8.6 Evaluation of User Simulations

Several metrics have been proposed to evaluate user simulations , e.g. (Ai and Litman, 2006; Georgila et al, 2006a; Schatzmann et al, 2005a; Scheffler and Young, 2001; Williams, 2007). See Pietquin and Hastie (2011) for a survey on metrics for the evaluation of user simulations. An established measure of dialogue similarity is based on the Kullback-Leibler (KL) divergence[8], as also used by e.g (Cuayáhuitl et al, 2005; Jung et al, 2009), which is defined as follows:

$$D_{KL}(P||Q) = \sum_{i=1}^{M} P(i) * log \frac{P(i)}{Q(i)} \tag{7.6}$$

This metric measures the divergence between distributions P and Q in a context with M responses. Ideally, the KL divergence between two similar distributions is close to zero.

KL allows us to compare the raw probabilities as observed in the original data with the probabilities generated by our user simulation models. We then compare the KL results of the cluster-based and the smoothed user simulation against a random model and a majority baseline, see Table 7.12. The random model is constructed by assigning equal frequency to all four actions, whereas the majority baseline always predicts the most frequent action in one context. The comparison against the random baseline tests the hypothesis that our user simulations are more consistent with the observed data than random behaviour. The majority baseline represents the hypothesis that our user simulation explores a significantly wider range of behaviour than the most frequent user action.

The user simulation models have a small divergence from the original data suggesting that they are good simulations for training and testing policies. The smoothed and the cluster-based model gain on average 5 times lower KL scores than the baselines. We therefore conclude that both simulations show consistent (i.e. better than random) as well as varying (i.e. better than the majority decision) behaviour.

As mentioned above, we want user simulations for policy training to allow more exploration, whereas for testing we want user simulations which are more realistic. We therefore choose to test with the smoothed model because its low KL score shows that it is closest to the data, and we use the cluster-based simulation for training.

User simulations:		Baselines:	
smoothed	cluster	random	majority
0.087	0.095	0.43	0.48

Table 7.12 Kullback-Leibler divergence scores for the different User Simulations

[8] Also known as information divergence, information gain, or relative entropy

Note that the KL divergence only measures consistency with respect to specific dialogue contexts. However, user simulations also need to be coherent with respect to the dialogue history and the current user goal. We therefore model the user's goal (i.e. the song s/he is looking for) similar to "agenda-based user models" (Schatzmann et al, 2007a,c). The user goal corresponds to a database entry, which is randomly chosen in the beginning of each dialogue. Every time the user simulation generates a speech act, the corresponding value is chosen from the goal record, dependent on the slot value the system was asking for.

Note that the KL divergence only measures consistency with respect to specific dialogue contexts. However, user simulations also need to be coherent with respect to the dialogue history and the current user goal. In the next Section we explain how we incorporate the current user goal in the simulated user behaviour.

7.8.7 Speech Act Realisation Dependent on the User Goal

The user speech acts generated by the user simulations are still on an abstract (semantic) level. We therefore model the user's goal (i.e. the song s/he is looking for) similar to "agenda-based user models" (Schatzmann et al, 2007a,c). The user goal corresponds to a database entry, which is randomly chosen in the beginning of each dialogue. Every time the user simulation generates a speech act, the corresponding value is chosen from the goal record, dependent on the slot value the system was asking for. In the following subsection we explain how the speech acts are realised with respect to the current user goal in more detail.

7.8.7.1 Over-Answering

Most current dialogue systems allow the user to provide more than one value within the same turn. This is called "over-answering" and is an aspect of "mixed-initiative" dialogue interaction. We model over-answering for our user simulations by realising the speech act `add` as "provide asked slot value" (`prvAsked`) and "provide two slot values" (`prvTwo`) following the frequencies observed in the data: in 75.12% of cases the user provides only one value, in 23.88% the user provides two values within one turn.

The final probabilities for the user simulations, after integrating noise and over-answering, are shown below. Table 7.13 presents the probabilities for the cluster-based user simulation. Table 7.14 shows the probabilities for the smoothed bi-gram model. In each cell, the first number represents the probability assigned by the statistical model, the two numbers below represent how the user actions are realised depending on noise and over-answering. Annotated examples for the user speech acts can be found in the Appendix A.1 and A.3.

system act/user reply	add prvAsked/prvTwo	repeat same/diff	y/n yes/no	change	OOV
explConf	20.1 15.3/4.8	51.6 36.1 /15.5	20.1 11.2/8.8	4.3	4.0
implConfAskASlot	43.7 33.2/10.5	35.9 25.1/10.8	12.2 8.5/3.7	4.2	4.0
askASlot	43.7 33.2/10.5	35.9 25.1/10.8	12.2 8.5/3.7	4.2	4.0
presentListSpeech	37.8 28.7/9.1	37.8 26.5/11.3	9.0 6.3/2.7	11.4	4.0
presentListMM	37.8 28.7/9.1	37.8 26.5/11.3	9.0 6.3/2.7	11.4	4.0

Table 7.13 Final probabilities for generating user actions from the cluster based model

system act/user reply	add prvAsked/prvTwo	repeat same/diff	y/n yes/no	change	OOV
explConf	10.1 7.7/2.4	21.2 14.8/6.4	54.56 38.2/16.4	10.1	4.0
implConf	40.7 31.0/ 9.7	32.3 22.6/9.7	15.7 11.0/4.7	7.3	4.0
askASlot	40.2 30.7/ 9.6	34.3 24.0/10.3	13.7 9.5/4.1	7.8	4.0
presentListSpeech	33.4 25.4/8.0	30.2 21.1/9.1	27.1 19.0/8.1	5.3	4.0
presentListMM	38.3 29.2/ 9.11	39.9 27.9/12.0	1.8 1.3/0.5	16.0	4.0

Table 7.14 Final probabilities for generating user actions from the smoothed model

7.8.7.2 User Goal Modelling

In Section 7.8.1 we argued that a user simulation should generate behaviour which is consistent with the current user goal. Previous work has addressed the problem in various ways. For example, by constraining the users' actions by paths through a graph-based dialogue model (Scheffler and Young, 2001), or by integrating task-based information into the state which conditions the user action (Georgila et al, 2006b; Pietquin, 2004). The draw-back of these previous approaches is that a substantial amount for training data is needed to learn user speech acts and goal-directed behaviour in a goal-directed manner.

In this book we introduce an approach which separates the user goal from learning. The user goal is represented as a record of state-value pairs, corresponding to a (randomly chosen) database entry. Every time the user simulation generates a speech act, the corresponding value is chosen from the goal record, as illustrated in the following example.

Let us assume that the user's goal is instantiated with the following database record:

$$\left[\textit{user goal:}\ \begin{bmatrix}\texttt{genre= Alternative}\\ \texttt{artist= Radiohead}\\ \texttt{album= In Rainbows}\\ \texttt{title= Videotape}\end{bmatrix}\right]$$

Let us also assume that the user has already provided the artist and the genre, and the genre is also confirmed. The next slot the system asks for is `album name` (the system follows the implemented default ordering of slots). The system next tries to implicitly confirm the artist and asks the user to provide a value for the slot `album`:

System: *"Searching for music by Radiohead. Which album?"*
`implConf(artist=Radiohead),askSlot(album=?)`

The possible user actions are instantiated with values from the goal record as in Figure 7.10. The statistical user model then chooses one of these actions according to the probability distributions.

$$\left[\begin{array}{ll}
\text{ADD:} & \begin{bmatrix}\texttt{prvAsked(album=In Rainbows)}\text{: e.g. \textit{``From the album OK Computer.''}}\\ \texttt{provideTwo(album=In Rainbows,title=Videotape)}\text{:}\\ \text{e.g. \textit{``Lucky from the album OK Computer.''}}\end{bmatrix}\\
\text{REPEAT:} & \begin{bmatrix}\texttt{confirmReprovideSame(artist=Radiohead)}\text{: \textit{``Yes, by Radiohead.''}}\\ \texttt{negateReprovideDifferent(artist=Red Hot Chili Peppers)}\text{:}\\ \text{\textit{``No, by the Chili Peppers.''}}\end{bmatrix}\\
\text{Y/N ANSWER:} & \begin{bmatrix}\texttt{yes-answer(artist=Radiohead)}\text{: \textit{Okay.}}\\ \texttt{no-answer}\star\texttt{(artist=Red Hot Chili Peppers)}\text{:}\\ \text{\textit{No, I want something else.}}\end{bmatrix}\\
\text{CHANGE:} & \begin{bmatrix}\texttt{provideOtherSlot(title=Videotape)}\text{: \textit{``The song Lucky.''}}\end{bmatrix}\\
\text{OOV:} & \begin{bmatrix}\texttt{silence}\end{bmatrix}
\end{array}\right]$$

Fig. 7.10 Example for context-dependent realisation of possible user speech acts

The goal instantiation of the user acts `no-answer` and `negateReprovide-Different` (marked by *) follows special rules. Both of these user actions aim to correct a value which is mis-recognised by the system. The (hypothesised) user goal therefore does not correspond to the represented user goal anymore. In order to model this misalignment, we change the record of the presented user goal by choosing another entry from the database. However, the database search is constrained by the slot values which were already confirmed by the user (as we assume that these are already representing the 'true' user goal). This simple mechanism of goal updating allows us to integrate grounding information in the user simulation without having to learn this behaviour from data (i.e. this method is suited for small data sets). For the given example, the entries in the possible subset are all database entries which fulfil the constraint `genre=Alternative`. Note that we do not aim to model (phonetic) word confusions, but we rather model the effects of mis-understanding on the interaction (as discussed in Section 7.7). Therefore it is possible to randomly choose between the subset of retrieved database entries, not having to consider phonetic similarity.

Note that similar technique was also introduced by Schatzmann et al (2007a,c) under the name "agenda-based models". The major advantage in the approach presented in this book is that we also model misalignment and grounding. We also do not need to learn the ordering of slot values, as we encode this information in the user speech acts (`provideAsked` vs. `provideOther/change`).

In addition to the task-based constraints, the user behaviour also follows some logical constraints, similar to the implemented preconditions for system actions (see Section 7.5.3). In particular, we constrain the possible user actions when there is only one hit in the database. In this case, the user simulation can only accept or reject the presented information. In the case where there is only one hit and all the slots are filled and confirmed, the user simulation can only accept (as we assume that for confirmed slots the chance to be mis-recognised is zero and therefore the presented information is identical to the user goal).

In sum, this Section presented methods to build user simulations from limited amounts of training data, as well as methods to evaluate and realise them with respect to the current user goal. One simulation is built for training and one for testing dialogue strategies. The user simulation which shows more variance is used for training, the user simulation which shows more consistency is used for testing.

So far, we have only evaluated individual simulated components (such as the user simulation). In the next Section we introduce the objective function (also known as the "reward" for RL) which is used to evaluate the overall dialogue strategy.

7.9 Reward and Objective Functions

7.9.1 Method and Related Work

In this book we use the two terms "reward function" and "objective function" with slightly different connotations. The term *reward function* refers to the function which specifies how the dialogue goal is implemented in RL. Recall the definition from Section 3.2: it assigns a scalar reward value which represents the degree to which a state or action is desirable, i.e. it defines a mapping $r(d,i)$ from a dialogue d and a position in that dialogue i to a reward value r. The reward function penalises actions or states which do not contribute to the goal, and rewards actions or states which do. The learner does not try to maximise the immediate reward (which is just a indication of the goal), but the final reward (which is a measure whether the goal was reached). Also see Chapter 3.2.

The term *objective function* is a mathematical term describing a function associated with a general optimisation problem which determines how good a solution is. It is also used in the context of RL-based dialogue strategy learning, where the term is used to stress the fact that for RL-based dialogue systems the same function can be used for strategy training and as well as for strategy evaluation (Paek, 2006; Walker, 2005).

We use the term "objective function" to describe the (abstract) overall goal/ objective of a dialogue. The term "reward function" is used to refer to the concrete implementation of how this goal is expressed. For example, if the objective of a dialogue is to be efficient, the reward function penalises long dialogues, for example by assigning a negative value of -1 to each turn, e.g. (Henderson et al, 2008). The same objective function is used for evaluation, i.e. the efficiency of the learned strategy will be compared against the efficiency of a baseline strategy, e.g. (Lemon et al, 2006a).

In contrast to other fields of computational linguistics and speech research which apply optimisation methods (such as speech recognition or spoken language understanding), the objective function for dialogue strategies is less clear. The ultimate goal when building dialogue systems is to satisfy the needs and preferences of real users. However, needs and preferences of real users are often hard to measure and to estimate. For example, users change their needs and preferences dependent on the context (Hu et al, 2007). Furthermore, it is nor clear how to encode/ translate this abstract goal into design principles (e.g. what is the equivalent of a "satisfied user" in terms of implemented system behaviour?). In the following we review dialogue design principles used by previous research, in particular we review different approaches for scoring multimodal presentation in information-seeking dialogue.

7.9.1.1 Scoring Information Presentation

In most multimodal dialogue systems the number of verbally and multi-modally presented items is scored with respect to some scoring function. In the following we review some "best practices" for information presentation as suggested by Human-Computer Interaction (HCI) literature, threshold-based strategies implemented in current information-seeking dialogue systems, and work based on RL.

There is a substantial amount of literature on HCI which recommends "best practises" for dialogue strategy design (see Chapter 2.2.2). The recommended strategies for information presentation are usually based on universal "human factors", i.e. lower cognitive processes such as perception, attention, and memory capacity. They give general guidelines such as "reduce short-term memory load" (Shneiderman, 1997), as well as very specific suggestions on how many items to present and when to present them. The recommended menu size for spoken output is 3 to 4 items, e.g. (Balogh et al, 2004; Weinschenk and Barker, 2000). For presenting lists on the screen the "magic number"[9] is 7 ± 2 items, e.g. (Dix et al, 1998; Herczeg, 1994;

[9] "The magic number" refers to a popular study by Miller (1956).

Larson, 2003). Furthermore, most of the HCI literature recommends to always use both modalities in parallel. This recommendation is based on the assumption that providing the same information in multiple ways will increase information reception and memorability (following the dual-encoding theory by Paivio (1990) and the Multiple Resource Theory by Wickens (2002)). These guidelines still seem to have a great impact on how information presentation is realised in current multimodal systems, e.g. (Cao and Nijholt, 2008; Landragin, 2008).

Information seeking dialogue strategies are often implemented as rule-based systems with hand-coded thresholds. We now discuss two in-car information seeking systems in particular. Varges et al (2006), for example, only use verbal output. For controlling information presentation they use the following thresholds: fewer than 3 retrieved database items are verbally enumerated, 3 to 8 items are verbally summarised, and for any number higher than 8 the system asks for more constraints. The speech-controlled in-car MP3 player by Forlines et al (2005) in contrast always uses multimodal output for every system turn, which may contain up to 10 items maximum.

Multimodal presentation is also addressed in the context of RL-based dialogue strategy learning, e.g. (Heeman, 2007; Levin et al, 2000). Heeman (2007) learns when to display search results using RL, where the reward function assigns increasing negative rewards in the following order: displaying less than 5 items receives 0 reward, 6-12 items gets mildly penalised, 13-30 is penalised more, and showing more than 30 items gets heavily penalised. Levin et al (2000) don't report the concrete numbers they use, but report that for a number of retrieved items N "[the cost] is zero for N smaller than a reasonable value, and increases rapidly thereafter, where N depends on the medium used to output information to the user (it is generally small for voice based communication, and higher for display)."

In sum, previous work scored information presentation according to hand-coded heuristics, which (at best) are based on "best practises" recommended by HCI literature. One contribution of this book is a more principled approach to model the scoring function for information presentation, in contrast to the heuristics used by previous research. We use this function as a reward to automatically optimise dialogue strategies. It is obvious that multimodal presentation should always depend on the available screen size, and media allocation should also depend on the cognitive load of the driver, as discussed in Chapter 3.4 (also see (Becker et al, 2006)). In this book we assume fixed screen size (namely the one used in the experiments), as well as a constant cognitive load for the driver. In the following we first review different methods of how the reward function is determined in previous work on RL-based dialogue strategy learning.

7.9.1.2 Setting the Reward Function for Dialogue Strategy Learning

A variety of techniques and measures are applied for setting the reward function for RL-based strategy learning. For model-based learning the reward can be directly read off from the data. For example, Singh et al (2002) employ a binary reward measuring task completion for the NJFun system. Or alternatively, user scores from the questionnaires can serve as a reward, e.g. (Spitters et al, 2007). That is, model-based RL has a realistic judgement associated with each dialogue in the training data.

For simulation based learning however, the reward function cannot be read off from the data, but needs to be explicitly set, as we only have simulated dialogue and no real users rating the system. Most current research manually constructs a reward function where the reward is represented as the weighted sum of task success, dialogue length penalty, and mis-recognition as a reward function, e.g. (Frampton and Lemon, 2006; Levin et al, 2000; Pietquin and Dutoit, 2006b; Prommer et al, 2006). Setting these features and their respective weights by hand is usually done in a similar fashion as iterative threshold setting (as described in Chapter 4.3): the dialogue designer manually tunes the values until it allows learning of the desired behaviour.[10] This is why the reward function is also called "the most hand-crafted aspect of Reinforcement Learning" (Paek, 2006).

However, it seems that setting a reward function by hand is non-trivial and the resulting strategies do not necessarily seem to reflect real user preferences.

Most user studies only report on improved dialogue performance measures (such as dialogue length), but user satisfaction scores do not significantly improve, e.g. (Lemon et al, 2006a; Prommer et al, 2006; Singh et al, 2002).

In this book we design a reward function that reflects real user preferences by learning a model from data using the PARADISE framework (see description Section 2.2.2). To the authors' knowledge only Walker (2000); Walker et al (1998a) and Henderson et al (2005, 2008) previously used a reward function obtained from data.

7.9.1.3 Discussion of the PARADISE Framework

PARADISE has become a "de-facto standard" (Möller et al, 2007) for evaluating spoken dialogue systems. However, various aspects of the PARADISE framework have been questioned by previous research: In general, models obtained by PARADISE only fit the data poorly ($R^2 \approx 0.4$) (Möller, 2005a; Möller et al, 2007). PARADISE employs linear regression, which is a form of Supervised Learning. For a supervised model the following issues can cause a low data fit: inadequate target variable, missing input features, lack of training data, and a wrong bias (Alpaydin, 2004). We discuss these factors in the following.

The target variable of the regression model is the subjective user rating which should be predicted. In the original framework the target variable is "User Satisfaction" which is constructed by summing over different scores as obtained from a user survey. However, Hone and Graham (2000) argue that the resulting number is meaningless, as taking the sum over different scores could only be justified on the basis

[10] For example, Prommer et al (2006) hand-tune their reward function as follows. It is known from the user questionnaires that user prefer a dialogues in a range of 4 to 6 system turns. In order to adequately configure the trade-offs between the features in the reward function they iteratively test and design, until the reward function allows to learn dialogues which are 4-6 turns long.

that all items measure the same underlying construct (which can be determined by a factor analysis). Thus, in recent work often only a single target variable is used, e.g. (Hajdinjak and Mihelic, 2006; Möller, 2005a,b; Möller et al, 2007; Skantze, 2007a).

Möller (2005a,b); Möller et al (2007) also experimented with many different input features. They find that from a large set of potential dialogue performance measures only few correlated (weakly) with user judgements. Thus, finding predictive factors is non-trivial. Furthermore, the selected input features also clearly depend on the chosen target variable. These results confirm that setting a reward function is non-trivial when done by hand. Möller et al (2007) also investigate the influence of the size of training data. Surprisingly, more training data only resulted in a very slightly increased model fit.

Another factor which can cause low model fit is a wrong model bias, i.e. the assumptions made by a certain machine learning technique do not match the nature of the problem as represented in the data. (This problem is also shortly discussed in (Engelbrecht and Möller, 2007).)

In this book we apply a modified version of the PARADISE framework and use it for reward modelling. We make the following changes in particular:

- We use a single target variable, namely Task Ease.
- We use a hierarchical reward model, reflecting the structure of the task for learning.
- For the information presentation phase we design a non-linear reward function.

7.9.2 Linear Regression for Information Acquisition

First of all, we construct the overall reward function using multiple linear regression as done by the standard PARADISE framework. In contrast to the standard PARADISE framework we set the target variable to be predicted to a single dimension, namely Task Ease. We choose Task Ease as target outcome for the following reasons. First, Task Ease is significantly correlated with most of the other questions from the WOZ user questionnaire (see Chapter 6.4.2). For example, Task Ease is significantly correlated with overall User Satisfaction (Spearman's $\rho = .840, p < .01$) and Future Use (Spearman's $\rho = .475, p < .01$).[11] Thus, Task Ease can be seen representative for other dimensions of user satisfaction. Furthermore, measures like Task Ease and Task Efficiency are most important for task oriented dialogue, according to the *principle of the least effort* (Clark, 1996) which says: "All things being equal, agents try to minimize their effort in doing what they intend to do". Finally, we intend our reward function to have a hierarchical structure, where the multimodal score of the presentation phase is a predictive factor for the overall reward. Taking Task Ease as

[11] Spearman's rank correlation coefficient is used to measure correlation between two variables at the ordinal level. Spearman's ρ is the the non-paramatric equivalent to Pearson product-moment coefficient.

the target variable allows us to construct such a model where the multimodal score is included in the overall regression model.

Task Ease is constructed by taking the average of two user ratings on a 5-point Likert scale. The ratings from the following two questions are averaged :[12]

1. The task was easy to solve. (Question 2)
2. I had *no* problems finding the information I wanted. (Question 6)

For model selection we used stepwise linear regression. This technique combines forward and backward feature selection. Only features which are significant predictors of the target variable are included in the model. Furthermore, we only choose input features which are not correlated with each other, (as two correlated variables explain the same variance in the model and thus hinders each other from being selected by step-wise regression). For example, dialogue duration (as measured in seconds) is positively correlated with dialogue length in turns (Pearson's correlation $r = .536, p < .01$). Therefore dialogue duration is not entered into the model.

The following dialogue performance measures are used as input feature to the stepwise regression model (see Section 7.4.3.1 for a more detailed description of these features):

- Dialogue length in turns (*dialogueLength*)
- Average corruption rate of the user input (*delHist*)
- Task completion (manually annotated, see (Blaylock et al, 2006))
- Multimodal score from user questionnaire (also see next Section).

The resulting model selects dialogue length, task completion, and the multimodal user score, as shown in Equation 7.7. Dialogue length is a strong negative predictor, while task completion and multimodal score are positive predictors of the perceived task ease.[13]

[12] We take the average of several questions to construct the Task Ease dimension, rather than only taking one single question. We do this in order to meet the assumptions made by the linear regression model. In linear regression input and target variables are assumed to be on an (at least) *interval* scale. The results from Likert-scales, however, are typically *ordinal*. By taking the average of several ordinal values we aim to approximate interval scale (see `http://en.wikipedia.org/wiki/Likert_scale` (4. January 2011) for a discussion).

[13] It is interesting to note, that the number of deleted ("mis-recognised") words did not influence the perceived Task Ease, where for most other studies the mean recognition score is one of the most prominent features, e.g. (Möller et al, 2007; Skantze, 2005; Walker et al, 2000; Williams and Young, 2004a). This might be an artefact of the fact that in a WOZ study the understanding of the human wizard was hardly affected by the corrupted input (only 30% of the corrupted utterances lead to a (noticeable) communication error, see Section 6.2.3). Similar results are reported by Hajdinjak and Mihelic (2006) when using WOZ data for PARADISE. Hajdinjak and Mihelic (2006) argue against including ASR-related features in the regression model, as it's influence on user ratings is so dominate that it hinders other relevant parameters to enter the regression model, and therefore it should be excluded altogether. We (partially) agree, as in a sense, it should be obvious to the designer that improving speech recognition quality improves user satisfaction. Thus, by including ASR measures into the regression not much would have been gained. On the other hand, ASR features are an important factor when optimising error handling strategies. For our task however, the PARADISE regression model as in Equation 7.7 contains all the relevant information.

$$TaskEase = -\mathbf{20.2} * dialogueLength + \mathbf{11.8} * taskCompletion + \mathbf{8.7} * multimodalScore; \quad (7.7)$$

The overall model fit is low ($R^2 = 0.144$, $R^2_{adjusted} = 0.123$).[14] Despite the low fit to the initial data, it generalises well to unseen events, as we will show in Section 8.5.2

7.9.3 Non-linear Rewards for Information Presentation

The results from feature selection now allow us to formulate the reward for the information presentation phase as a simple two-way relationship between two variables. In Section 7.4.3.2 we found that the wizards' information presentation strategy is based on the number of retrieved items from the database (the `DB` feature). In order to design the reward function for information presentation we relate the number of presented DB items (independent variable) to the multimodal score from the user questionnaires (dependent variable). The multimodal score is measured by taking the average of the user ratings for the following questions (translated from German):

1. I liked the combination of information being displayed on the screen and presented verbally.
2. Switching between modes did *not* distract me.
3. The displayed lists and tables contained on average the right amount of information.
4. The information presented verbally was easy to remember.

The questions are rated on 5 and 3-point Likert Scales, and the scores are normalised for taking the average.

For model selection we use curve fitting (also known as non-linear regression). Curve fitting is used when the relationship between two variables can also be non-linear. It allows us to choose the model/curve which is closest to the data. In contrast to linear regression, curve fitting does not assume a linear inductive bias, but it selects an appropriate inductive bias by function interpolation. The simplest model with the best data fit is the one which gets selected (due to Occam's razor). The following models are fitted to the data – first relating verbally presented items to the multimodal score and then relating items presented on the screen to the multimodal score:

- Linear ($Y = b0 + (b1 \times t)$)
- Logarithmic ($Y = b0 + (b1 \times ln(t))$)
- Inverse ($Y = b0 + \frac{b1}{t}$)

[14] R^2 indicates how much *variance* is explained by the model; in this case 14.4%. Adjusted R^2 is a modification of R^2 that adjusts for the number of explanatory terms in a model.

- Quadratic ($Y = b0 + (b1 \times t) + (b2 \times t^2)$)
- Cubic ($Y = b0 + (b1 \times t) + (b2 \times t^2) + (b3 \times t^3)$)
- Power ($Y = b0 \times (t^{b1})$ or $ln(Y) = ln(b0) + (b1 \times ln(t))$)
- Compound ($Y = b0 \times (b1^t)$ or $ln(Y) = ln(b0) + (ln(b1) \times t)$)
- S-curve ($Y = e^{(b0+(\frac{b1}{t}))}$ or $ln(Y) = b0 + (\frac{b1}{t})$)
- Logistic ($Y = 1/(\frac{1}{u} + (b0 \times (b1^t)))$ or $ln(\frac{1}{y} - \frac{1}{u}) = ln(b0) + (ln(b1) \times t)$ where u is the upper boundary value)
- Growth ($Y = e^{(b0+(b1 \times t))}$ or $ln(Y) = b0 + (b1 \times t)$)
- Exponential ($Y = b0 \times (e^{b1 \times t})$ or $ln(Y) = ln(b0) + (b1 \times t)$)

The best fitting model for relating the verbally presented items is a linear decreasing line ($R^2 = 0.24, R^2_{adjusted} = 0.21$) assigning positive values to all strategies which present fewer than 4 items verbally, and a negative value to all strategies which present 4 or more items verbally (see Equation 7.8 where x is the number of presented database items, and see Figure 7.11). Note, that in the WOZ experiments the wizards never present more than 3 items verbally. For training the policy (where unseen state spaces also need to be explored) we extrapolate the fitted line to model unobserved behaviour.

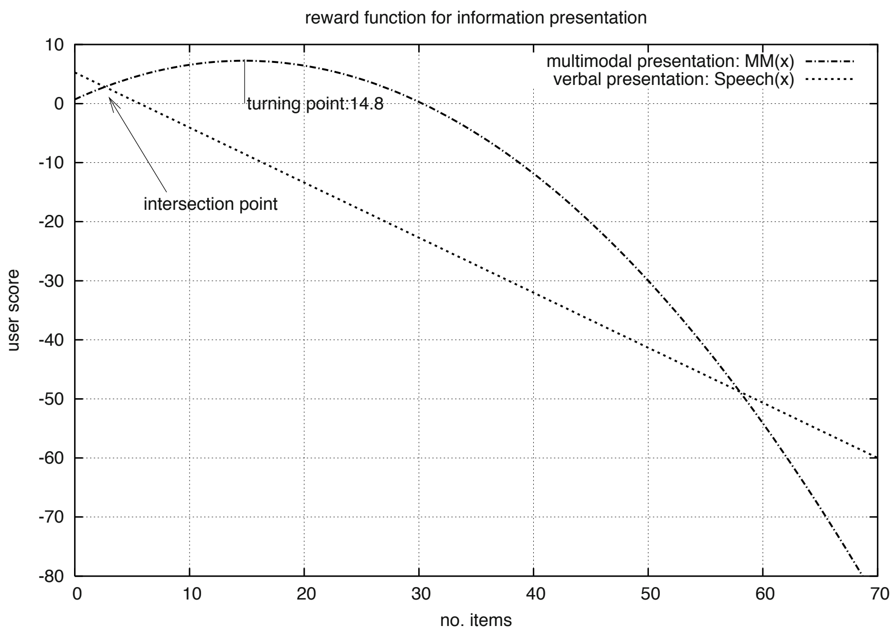

Fig. 7.11 Evaluation functions relating number of items presented in different modalities to multimodal score

The best fit for the wizards' multimodal strategy is a quadratic model ($R^2 = 0.12$, $R^2_{adjusted} = 0.103$), (see Equation 7.8 and Figure 7.11). The resulting model assigns a maximal score to a strategy displaying 14.8 items (curve inflection point). For items higher than 30 it assigns negative values (intersections with x-axis at $x_1 = -0.1, x_2 = 29.63$). The two reward functions intersect at $x_1 = 2.62, x_2 = 57.98$.

$$multimodalScore = + \begin{cases} Verbal(x) = 5.259 + (-0.932 \times x); \\ \qquad\qquad\qquad\qquad \text{for } x : [1, +\inf] \\ MM(x) = 0.705 + 0.886 \times x - 0.03 \times x^2; \\ \qquad\qquad\qquad\qquad \text{for } x : [1, +\inf] \end{cases} \tag{7.8}$$

Again, the fit to the initial data is rather low (other studies report on R^2 up to 0.47). Nevertheless, these models make useful predictions, as we will show in Section 8.5.2.

7.9.4 Final Reward

Finally, we integrate the obtained models into one (hierarchical) reward function for learning. The final reward is calculated as in Equation 7.9a – 7.9d. The overall reward, as calculated at the end of each dialogue, corresponds to the reward for information acquisition, see Equation 7.9a. It is computed as follows. After each system turn the the dialogue length receives a penalty of (-1), see Equation 7.9b. The task completion is calculated according to the noise model, see Equation 7.9c, where Pc is the probability of a confirmed slot being correct, and Pf is the probability of a filled slot being correct, where C and F are the number of confirmed slots and filled (but not confirmed) slots respectively. A more detailed explanation of how task completion is calculated can be found in Section 4.2.5.

$$reward = 20.2 \times dialogueLength + 11.8 \times taskCompletion + \\ 8.7 \times multimodalScore; \tag{7.9a}$$

$$dialogueLength = numberOfTurns \times (-1); \tag{7.9b}$$

$$taskCompletion = 10 \times (Pc)^C \times (Pf)^F; \tag{7.9c}$$

$$multimodalScore = Verbal(x) + MM(x); \tag{7.9d}$$

We illustrate how the final reward is calculated in the following examples. In the dialogue in Appendix A.2, Table A.3 the Supervised Learning policy receives a final reward of -18701.43. Following the equations above, this is calculated as follows. The dialogue is 7 turns long. Thus the length penalty is $20.2 \times [7 \times (-1)] = -141.4$. All the slots are filled and confirmed. Thus the task completion is $11.8 \times [10 \times (1)^4 \times (0.3)^0] = 118$. In the dialogue 142 items are presented multi-modally and 1 item is presented verbally. Thus the multimodal score is $8.7 \times [4.33 - 2151.23] =$

-18678.03, where $Verbal(3) = 5.259 + (-0.932 \times 3 = 4.33)$; $MM(142) = 0.705 + 0.886 * 142 - 0.03 * 142^2 = -2151.23$.

An example of a dialogue strategy which is rated highly under this reward function can be found in Appendix A.3, Table A.8. Here the user immediately asks for a song called "Polly", for which two alternatives are found in the database. These alternatives are presented verbally ($Verbal(2) = 3.4$) and the user selects one of the options ($Verbal(1) = 4.33$). This results in a dialogue which is 3 turns long ($lengthPenalty = 20.2 \times (-3) = -60.6$) and the task is fully completed ($complVal = 118$). The final reward for this dialogue is 124.6, which is close to the maximum possible reward which is 158.10. This maximum reward is achieved when the user immediately provides a slot value for which 15 items can be retrieved in the database. The system immediately presents multimodally (MM(15)=7.25) and the user selects an item. Then, the system presents verbally (Verbal(1)=4.33), and the user confirms. This "optimal" interaction results in a dialogue which is 3 turns long ($lengthPenalty = -60.6$), where all the filled slots are confirmed ($taskCompletion = 118$). The learned policy in Appendix A.3, Table A.7, for example, almost achieves the maximal possible reward.

To the authors' knowledge the work presented in this book is the first to learn with non-linear reward functions. This also imposes new challenges for the RL framework. In the next Section we describe prerequisites for learning with non-linear reward functions.

7.10 State-Space Discretisation

We use linear function approximation in order to learn with large state-action spaces. Linear function approximation learns linear estimates for expected reward values of actions in states represented as feature vectors. This is inconsistent with the idea of non-linear reward functions (as introduced in the previous section). We therefore quantise the state space for information presentation. We partition the database feature into 3 bins, taking the first intersection point between verbal and multimodal reward and the turning point of the multimodal function as discretisation boundaries. Previous work on learning with large databases commonly quantises the database feature in order to learn with large state spaces using manual heuristics, e.g. Pietquin (2006) use 2 bins, Levin et al (2000) use 4 bins, and Heeman (2007) 5 bins. Our quantisation technique is more principled as it reflects user preferences for multimodal output. Furthermore, in previous work database items were not only quantised in the state-space, but also in the reward function, resulting in a direct mapping between quantised retrieved items and discrete reward values, whereas our reward function still operates on the continuous values. In addition, the decision of *when* to present a list (information acquisition phase) is still based on continuous DB values. In future work we plan to engineer new state features in order to learn with non-linear rewards while the state space is still continuous. A continuous representation of the state space allows learning of more fine-grained local trade-offs between the parameters, as demonstrated in (Rieser and Lemon, 2008b).

7.11 Learning Experiments

We now train and test the multimodal presentation strategies by interacting with the simulated learning environment. We first train the policy using the SHARSHA algorithm (Section 7.11.1). We then test the strategy using a different user simulation. We compare the performance of the RL-based policy against the supervised baseline (Section 7.11.2). And finally, we implement the learned strategies in a table-look-up (Section 7.11.4). The look-up table can be found in Appendix B.

7.11.1 Training with SHARSHA

The policy is trained by interacting with the cluster-based user simulation which is predicted to allow more exploration, see Section 7.8.6. We use the REALL-DUDE environment (Lemon et al, 2006c) for learning (see Section 7.5.2). REALL combines Reactive Planning and Hierarchical RL (Shapiro, 2001), as explained in Section 7.5.3. Hierarchical RL is implemented using the SHARSHA algorithm (for State Hierarchy, Action, Reward, State Hierarchy, Action) (Langley et al, 2004; Shapiro, 2001; Shapiro and Langley, 2002). The SHARSHA algorithm adds hierarchical structure to the well known SARSA algorithm (see explanation in Section 3.2.2.2).

For learning we use the following parameters (for a description of these parameters see Section 4.4.1):

- numberOfCycles = 180k
- learning rate $\alpha = 0.2$
- discount rate $\gamma = 0.95$
- eligibility trace parameter λ=0.9
- exploration halving time ε= $\frac{1}{6} \times numberOfCycles = 30k$

The policy is trained over 180k system cycles, which results in about 20k simulated dialogues. The two figures in Table 7.15 show a training run, where the graph on top shows the whole run for over 20k dialogues. The picture below zooms-in to the area between 5k and 13k training cycles, where the learner converges towards the optimal policy. The policy learns to reduce dialog length (black, -1 per turn) and database hits presented multimodal (pink, +1 per item) and presented verbally (blue, +1 per item) while getting a high task completion (green). Average dialogue reward, computed over windows of 50 dialogues, is shown in red.

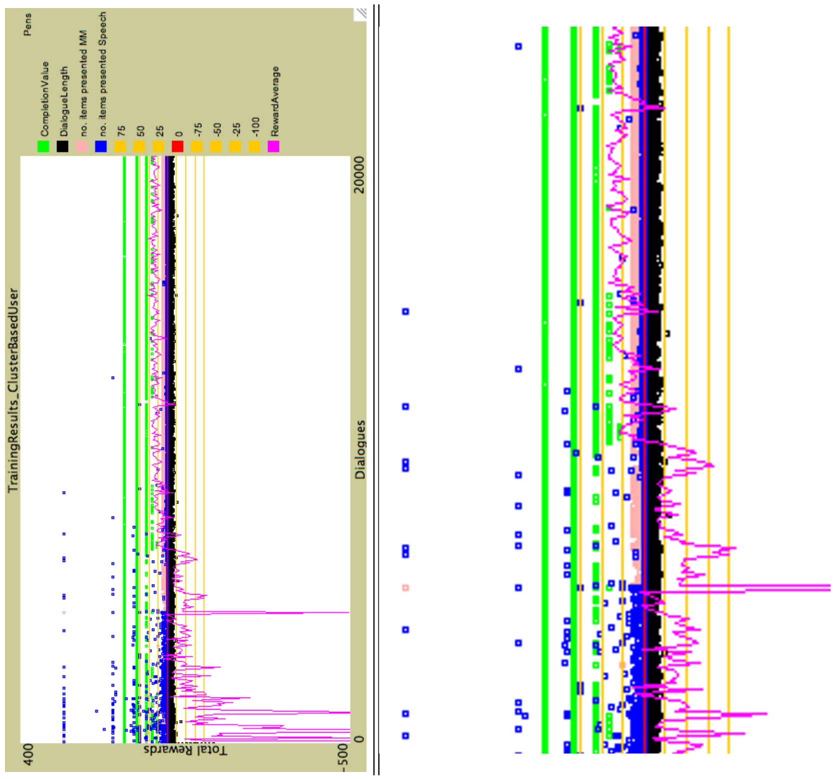

Table 7.15 Learning curve for training: whole training run (top), zoom-in between 5k and 13k dialogues (bottom)

7.11.2 Results for Testing in Simulation

We test the RL-based and the SL wizard baseline policy, as shown in Listing 2, which allows us to measure relative improvement over the training data. We run 500 test dialogues with the smoothed user simulation, as described in Section 7.8.5, so that we are not training and testing on the same simulation. We then compare quantitative dialogue measures by performing a paired t-test. In particular, we compare mean values of the final rewards, number of filled and confirmed slots, dialogue length, and items presented multimodally (`MM items`) and items presented verbally (`verbal items`).

RL performs significantly better ($p < .001$) than the baseline strategy. The RL-based policy gains on average 50-times more reward when tested in simulation. It results in significantly shorter dialogues, where significantly fewer slots get confirmed, while about the same number of slots get filled. The only non-significant difference is the number of items presented verbally, where both the RL and the average wizard strategy settled on a threshold of fewer than 4 items. The mean performance measures for simulation-based testing are shown in Table 7.16.

Measure	**SL wizard baseline**	**RL Strategy**
av. turns	8.42($\pm$3.04)	5.9($\pm$2.4)***
av. speech items	1.04($\pm$.2)	1.1($\pm$.3)
av. MM items	61.37($\pm$82.5)	11.2($\pm$2.4)***
av. reward	-1741.3($\pm$566.2)	44.06($\pm$51.5)***

Table 7.16 Comparison of results for SL wizard and RL-based strategies in simulation; *** denotes significant difference between SL and RL at $p < .001$ (with standard deviation $\pm$)

The major strength of the RL policy is that it learns to keep the dialogues reasonably short (on average 5.9 system turns for RL versus 8.4 turns for the SL wizard) by presenting lists as soon as the number of retrieved items is within tolerance range for the respective modality (as reflected in the reward function). The wizard strategy in contrast has not learned the right timing nor an upper bound for displaying items on the screen (note that the distribution for `MM items` is highly positively skewed with a maximum of 283 items being displayed). See example dialogues in Appendix A.2.

The results show that simulation-based RL with an environment bootstrapped from WOZ data allows learning of robust strategies which significantly outperform the strategies learned by SL from the original data set. This confirms our hypothesis that simulation-based RL allows us to find optimal policies which are not easily discoverable (by Supervised Learning) in the original data.

Furthermore, RL allows us to provide additional information about user preferences in the reward function, whereas SL simply mimics the data. In addition, RL is based on delayed rewards, i.e. the optimisation of a final goal. For dialogue systems we often have measures indicating how successful and/or satisfying the overall performance of a strategy was, but it is hard to tell how exactly things should have been done in a specific situation. This is what makes RL specifically attractive for dialogue strategy learning. In Chapter 8 we test the learned strategy with real users. In the next Section we give a qualitative description of the learned strategy.

7.11.3 Qualitative Strategy Description

We now give an example-based description of the RL-based and the wizard baseline strategy. We show that both policies originate from the same WOZ data set. We also give examples showing how the additional information about user preferences (as obtained from the user ratings and modelled in the reward function) helps RL to rule-out the weak strategies in the data, whereas SL simply mimics the data. The example dialogues can be found in Appendix A.1 and A.2.

We first describe different strategies for *information presentation*. In the WOZ experiments the wizards display very long lists on the screen. In the WOZ dialogue in Table A.1, for example, the wizard displays a list with 142 items (Wizard: "*The song is retrieved. There are 142 different versions.*" [Table with 142 entries is shown on the screen.]). This presentation strategy was rated low by the user (timing:2, mm-Presentation:1, verbalPresentation:3). For RL-based strategy learning, these ratings are used to estimate the reward function with the PARADISE framework, as explained in Section 7.9.3. The SL policy in contrast only mimics the wizard's behaviour. In the simulated dialogue in Example 7.1 and Table A.3, the SL policy also displays 142 items to the user (immediately after the user provided the first constraint). Here the number of current search results is shown by "db:".

Example 7.1.

```
SL policy:   greet (db:438)
sim.User: prvAsked-->genre(Rock)
SL policy: implConf(genre),AskASlot(artist) (db:142)
sim.User: confirmAndReprvSame-->genre(Rock)
SL policy:  present[mm] -->genre (db:142)
sim.User: change-->artist(Jimi Hendrix)
SL policy: implConfAskASlot-->artist, album
sim.User: prvAsked-->album(Electric Ladyland)}
SL policy: implConfAskASlot-->artist, album (db:4)
sim.User: prvAsked-->song-title(Voodoo Child)
SL policy: explicitConfirm-->song-title (db:1)
sim.User: confirmAndReprvSame-->song-title(Voodoo Child)
SL policy:  present[verbal] (db:1)
sim.User: yes-answer-->yes(yes)
```

The RL policy in contrast learns to wait and first gather more constraints until an adequate number of items is retrieved. In the simulated dialogue in Example 7.2 and Table A.4 the RL policy displays 14 items, which according to the reward function, gains the highest multimodal reward.

Example 7.2.

```
RL policy: greet (db: 438)
sim.User:  prvAsked-->artist(Nirvana)
RL policy: implConf(artist)AskASlot(album) (db:26)
sim.User: prvAsked-->album(MTV Unplugged)
RL policy: present[mm] -->artist, album (db:14)
sim.User: click-->song-title(On a Plain)
RL policy: present[verbal] (db:1)
sim.User: yes-answer-->yes(yes)}
```

We now describe different strategies for *information acquisition* and show how the RL and SL policies both originate from the WOZ data set.

In the WOZ study the wizards sometimes fail to present results early enough in the information acquisition phase. In the WOZ dialogue Table A.2 in Appendix A.1, for example, the wizard does not present the retrieved items until the user explicitly asks to see the list (Wizard: "*I am searching... I found four songs with only 'Smile' as the title.*"; User: "*Very well then, why don't you show it?*"). The wizard then shows a list of 4 items, which the user rates with the maximal score (mmPresentation). Task Ease and presentation timing, however, are rated low ($TaskEase : 2, timing : 3$).

The SL policy in the simulated dialogue in Example 7.1 (also see Table A.3 in Appendix A.2) replicates this behaviour and fails to present the retrieved items over several turns as well, even though the number of retrieved database items is already fairly low in the beginning (4 items are retrieved after the third turn).

The RL policy in contrast learns to present as soon as the number of retrieved items is adequate for verbal or multimodal presentation. In the simulated dialogue in Example 7.2 (also see Table A.4 in Appendix A.2), for example, the RL policy displays 14 items on the screen in the third turn of the dialogue.

In the simulated interaction in Example 7.3 (also see Table A.5) the RL policy presents 2 items verbally in the third turn. This action choice corresponds to the wizard's behaviour in Table A.1 in Appendix A.1 (Wizard: "*The song exists by 'Eric Clapton' or by 'Eric & the Dominos'.*"), where the verbal presentation was rated high by the user ($presentVerbal : 3$).

Example 7.3.

```
RL policy: greet (db: 438)
sim.User: prvAsked-->artist(Bob Dylan)
RL policy:  implConf(artist)AskASlot(genre) (db:56)
sim.User: change-->song-title(Hurricane)
RL policy:  present[verbal] -->album1,album2 (db:2)
sim.User:  prvAsked-->album(Desire)
RL policy:  present[verbal] -->artist, album, song-title (db:1)
sim.User: yes-answer-->yes(yes)
```

In sum, the dialogue examples demonstrate how information from the WOZ data is used by the different learning mechanisms: One major advantage of RL is that it allows us to provide additional information about user preferences in the reward function, whereas SL simply mimics the data.

7.11.4 Strategy Implementation

The RL strategy has been operating in simulation so far. For strategy testing in the simulated environment, we set the RL agent's learning rate to be zero. The Dialogue Manager calls the RL agent each time a strategy decision is needed. The same approach can be taken when integrating the learned strategy into a working dialogue system, e.g. by 'wrapping' REALL as an OAA agent. A similar method has been used by e.g. (Lemon et al, 2006a) However, this method has been criticised as being a "black box which tells the system what to do" (Paek, 2006). Paek claims that for this reason, RL-based dialogue design is less attractive for industry, as it takes away the control from the system designer and makes it hard to change the strategy if required.

An alternative method for strategy implementation is to transfer the learned strategy into a table, where the Dialogue Manager can look-up which action to take in which state. The look-up table for the learned strategy described in this book can be found in Appendix B, which indicates the action with the highest expected reward for each state. This method has also been used in previous research, e.g. (Singh et al, 2002; Walker, 2000).

Both methods are appropriate under different circumstances. Implementing the strategy as an RL agent has the advantage that it can handle large state-action spaces (given that the implemented algorithm is able to do so). Furthermore, the system has the option to learn online (i.e. while it is employed). Online learning has the major advantages that it can adapt to the individual user and also can constantly adapt to changing environmental conditions. However this kind of implementation faces the criticism of being a 'black box'. The look-up table implementation, in contrast, is a clear and detailed description of what was learned, which allows the dialogue designer to understand and, if needed, change the policy. However, a table-look-up easily becomes intractable for large state-action spaces.

For the given strategy learning problem there are $4^{112,128} + 2^{2^3}$ *theoretically* possible policies (see Section 7.5), the number of policies which are *practically* possible in the concrete application environment is much lower. This is due to the following constraints. First, preconditions (which are implemented as ISU rules) prohibit state-action combinations which are inconsistent with the system logic (see Section 7.5.4). In addition, the possible values for the number of retrieved items (`DB`) are limited by the semantic relations between the database and the provided constraints, i.e. there are restrictions by the given "ontology". In our database, for example, there are only 3 genres (see Section 7.2). That is, if the only constraint provided by the user is 'genre' there are only 3 possible partitions of the database, and the DB feature can only take the values $DB = \{175/142/34\}$, i.e. for "Blues" 175 items are retrieved, for "Rock" 142, and for "Alternative" 34.[15] The same is true for the other slots values and their combinations. For example, in our database an album can contain a range of 4 to14 songs, and a song can be maximally quadruply ambiguous

[15] Note that for our simulated retrieval model in Section 4.2.2 did not encode such domain-specific constraints and therefore a larger policy space had to be explored.

(see Section 7.2). Hence, the retrieval behaviour of the underlying database does influence strategy learning (as also shown in Chapter 4.2.2 by simulating different retrieval methods).

The constraints from the database, together with the ISU update rules, leave us with a restricted set of possible state-action combinations, which is sufficiently small for a table-look-up implementation. As we (currently) do not learn online, we prefer this approach as it is easy to inspect and understand for the system developer as well as for the reader of this book . In total the learned strategy has 371 state action pairs as shown in Appendix B. This list is constructed by running the DUDE-REALL system in test mode (i.e. with zero learning and exploration rate). We generate a corpus of over 5k simulated dialogues, where we also log the corresponding system state. From this corpus we extract all possible state-action combinations. After ordering the state-action space (starting with no slots filled), one can observe stable patterns, as shown in the table in Appendix B.

On an abstract level the state-action mappings can be described as in Algorithm 3. Of course, there are several context-specific exceptions from the general rule set. For example, the preferred confirmation act is an `implicitConfirm`. An exception is the case where the user has already provided the song name (slot4). The system then explicitly confirms rather than implicitly confirms, in order to end the dialogue as soon as possible. Note that strategies which are sensitive to the different semantic qualities of slots types can only be learned with a binary representation of slot state features (as discussed in Section 7.4.2).

Furthermore, if there are too many slots which are filled but not confirmed, the system tries to get the user to confirm first, although the number of retrieved items may be already small enough to present.

Algorithm 3 $Highlevel - Description - LearnedStrategy$

Require: An integer $DB \geq 1$.
DB $\Leftarrow$ number of retrieved database hits
repeat
    gather constraints by implConf whenever allowed by the system logic
until $DB \leq medium$
if $DB = low$ **then**
    return `presentSpeech`
else if $DB = med$ **then**
    return `presentMM`
end if

7.11.5 Discussion and Error Analysis

In this book we have formalised the dialogue strategy problem as being task-independent, i.e. the user simulation as well as the system employ a separate task model and discourse model. Only the discourse behaviour is learned, whereas the task model is set by hand following a default ordering (see Section 7.5.4). These types of model separation for Dialogue Management gained popularity in the dialogue management community (Bohus and Rudnicky, 2003; Lemon and Liu, 2006; Lemon et al, 2002; Xu and Rudnicky, 2000), as well as for user simulation (Schatzmann et al, 2007a,c). In the following we show the advantages and disadvantages of this separation for strategy learning.

7.11.5.1 Consistent User Behaviour

In this book we introduce a separate user goal as an extension to simple bi-gram models. The bi-gram model is learned from data and generates discourse behaviour, while the user goal is defined as a record with default ordering (see Section 7.8.7). We illustrate the advantages of this separation below.

Previous work has criticised bi-gram to be "without memory" (see Section 7.8.1), e.g. (Pietquin and Dutoit, 2006b; Schatzmann et al, 2005b).[16] However, introducing a separate model of user goal allows to learn behaviour which is consistent with the user goal as well as the dialogue history, as illustrated in the following example (Note that all the following examples are taken from the corpus generated in the simulated test interaction described in Section 7.11.2):

Example 7.4.
```
 [...]
System1: implConfAskASlot-->album, song (db:11)
User1:  provide_info-->song(It Aint Me Babe)
        filled [genre, artist, album, song],
        confirmed [genre, artist, album]
System2:presentList[speech]-->genre, artist, album, song (db:1)
User2:  no_answer-->no(no)
        filled [genre, artist, album],
        confirmed [genre, artist, album]
System3: presentList[mm] -->genre, artist, album (db:11)
User3:  provide_info-->song(All I Really Want to Do)
        filled [genre, artist, album, song],
        confirmed [genre, artist, album]
System4:presentList[speech]-->genre, artist, album, song (db:1)
User4:  yes_answer-->yes(yes)
        filled [genre, artist, album, song],
        confirmed [genre, artist, album]
```

In Example 7.4 the user first rejects (`User2`) a single item presented verbally (`System2`). As a consequence, the system un-fills the last provided slot (in this case the song title, see `User1`), and queries the database with the reduced set of

[16] Note that our cluster-based user model (theoretically) allows to include dialogue history features, which enables this model to produce history-sensitive behaviour. In related work we show that cluster-based models including history features outperform models which do not (Rieser and Lemon, 2006a). In this book however, we the applied cluster-based model does not include any history features.

constraints. Furthermore, the user goal gets updated by choosing another song from the subset defined by the confirmed slots (album and artist), also see Section 7.8.7. In `User3` the user provides a different song meeting these constraints. The system then present this item verbally. In this case, the user simulation can do nothing but to accept the presented, as we implemented the constraint that if all slots are filled and confirmed the presented item has to be identical with the user goal. (Assuming that confirmed slots are correct with a 100% likelihood (also see Section 4.2.5).) In sum, the user simulation shows complex goal and history consistent behaviour, by using the implemented constraints of the user goal, together with the learned discourse behaviour from the bi-gram model. This approach has the advantage that the discourse behaviour can be kept relatively simple (and thus can be learned from small data sets), while the task behaviour is controlled by an external model.

7.11.5.2 Learning Optimal Slot Ordering

Despite its advantages for learning from small data sets, the separation of task and discourse model also introduces new challenges, which we discuss as interesting directions for future research.

At the moment the task model uses a simple default ordering of slots, which is inspired by related work on music player applications for in-car, e.g. (Forlines et al, 2005; Schulz and Donker, 2006; Varges et al, 2006; Wang et al, 2005), also see Section 7.5.4. However, this hand-crafted default ordering leads to unnatural (or at least un-logical) behaviour where user simulation or the system consider redundant information to be important.

Example 7.5.
```
System1:  greet--> (db: 438.0)
User1:    provide_info-->album(I Might Be Wrong)
         filled [album], confirmed []
System2:  implConfAskASlot-->album, genre (db: 8)
User2:    provide_info-->artist(Radiohead)
         filled [album, artist], confirmed [album]
System3:  presentList[mm] -->album, artist (db: 8)
User3:    provide_info-->genre(Indie Rock)
         filled [album, artist, genre], confirmed [album]
System4:  implConfAskASlot-->artist, song_title (db: 8)
            [...]
```

In Example 7.5 the user simulation has already specified album (`User1`) and artist (`User2`). The user then provides the genre (`User3`) where genre is redundant, as only one possible genre for this artist exists. Task-Consistent behaviour would be for the user to choose an item from the list presented multimodally (`System3`).

Note that Example 7.5 also illustrates that the RL-based strategy applies flexible “thresholds” (thus Algorithm 3 is really just a very general simplification of what was learned). In Example 7.5 the system keeps on asking for new constraints, hoping that the user will provide a song title and thus a list can be presented verbally (which gains a high immediate reward). It therefore delays the change to present a list multimodally, although a small number of possible candidates is retrieved right at the beginning of the dialogue.

Furthermore, the system also asks for redundant information. In Example 7.6 the system asks for genre (`System2`), where (in our database) the genre is already defined by the artist. The same is true for `System2` in Example 7.5. Note that other studies have identified the slot type "genre" to be in general problematic for this task (Schulz and Donker, 2006; Wang et al, 2005).

Example 7.6.

```
System1:  greet--> (db: 438.0)
User1:    provide_info-->album(Bob Dylan)
    State: filled [slot2], confirmed []
System2:  implConfAskASlot-->artist, genre (db: 92)
User2:    provide_info-->genre(Blues)
    State: filled [slot1,slot2], confirmed [slot2]
System3:  implConfAskASlot-->genre, album(db: 92)
User3:    provide_info-->album(Desire)
    State: filled [slot1,slot2,slot3], confirmed [slot1,slot2]
System4:  presentList[mm] -->genre, album, artist (db: 9)
             [...]
```

Related work has proposed the following solutions for encoding task-related information.

For learning task-consistent user behaviour, (Schatzmann et al, 2007c) extends the agenda-based models introduced in (Schatzmann et al, 2007a) with an agenda which is learned from data. The agenda (or user state) is treated as a hidden variable which has the advantage that the amount of required annotated training data can be reduced. This approach is a promising direction for future research.

For learning task-consistent system behaviour, the slot ordering can be included in the RL task, i.e. which slot type to ask next can be included in the action set, e.g. (Pietquin and Beaufort, 2005; Thomson et al, 2008; Williams, 2006; Young, 2006). However, this approach does not maintain a separate task model, but merges discourse and task behaviour. Furthermore, most of the previous work on learning optimal slot ordering with RL primarily focused on improving ASR quality, and did not explicitly target task-consitent behaviour.

Most of the work which maintains a separate task model, manually specifies the ordering of the task, e.g. (Bohus and Rudnicky, 2003; Xu and Rudnicky, 2000). However, recent work suggests some more principled approaches to determine which slot to ask next. Becker et al (2006); Polifroni and Walker (2006) choose the next slot according to the expected information gain with respect to the structure of the database, where Polifroni and Walker (2006) use decision trees derived from data; Becker et al (2006) use hand-coded rules. Other recent work learns the task model from human-human data (Bangalore et al, 2006).

An interesting direction for future research is to determine how to best integrate task and discourse behaviour into the RL framework.

7.12 Summary

In this Chapter we learned an optimal strategy for information seeking dialogue systems using simulation-based RL, where the simulated environment is obtained from limited amounts of WOZ data.

We address the complex and challenging problem of *how many* database search results to present to the user, and *when* to present them, and *which modality* to use for presentation, given the competing trade-offs between the number of results (large lists are difficult for users to process especially when only presented verbally), the length of the dialogue (long dialogues are tiring, but collecting more information can result in more precise results), and the noise in the speech recognition environment (in high noise conditions accurate information is difficult to obtain).

To train and test such a policy in a efficient but accurate manner, we obtain a simulated learning environment from WOZ data, where all the simulated components are constructed using data-driven methods. The overall framework is summarised in Figure 7.12.

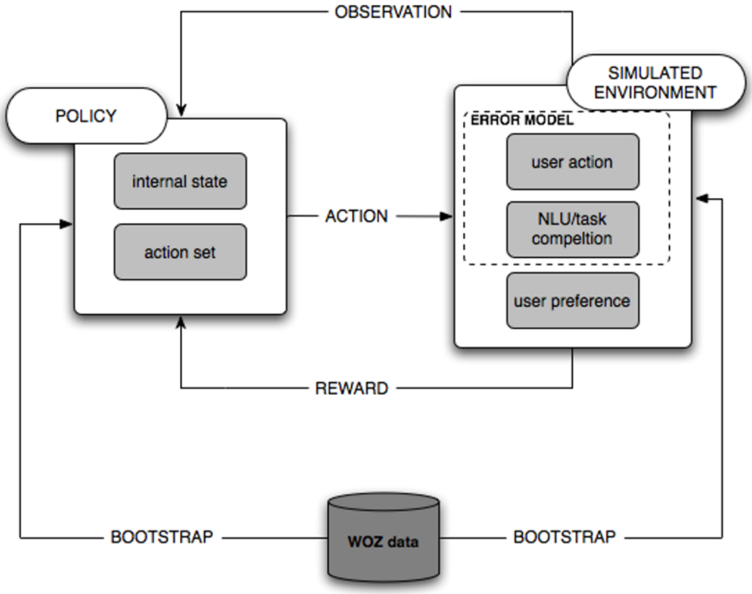

Fig. 7.12 Overall framework for RL in a simulated environment obtained from WOZ data

In particular, we create the action set by exploring the actions taken by the wizards, and we used automatic feature selection techniques in order to define the state space. We presented a method for simulating channel noise if training data is sparse, and we also introduce methods to build and evaluate user simulation from small data sets. The applied reward function is obtained using a modified version of the PARADISE framework.

We then trained and tested a RL-based policy by interaction with the simulated environments. We compare the results against a supervised baseline policy, reflecting the "average" wizard behaviour. This comparison allows us to measure the relative improvements of the RL-based policies over the strategies obtained from the initial data set. The RL policy significantly outperforms ($p < .001$) the supervised baseline strategy, gaining on average 50-times more reward when tested in simulation. One main advantage for Reinforcement Learning over Supervised Learning in this context is that it allows us to incorporate user feedback in the reward function, and therefore allows learning of strategies which reflect user preferences.

We also described the RL-based policy. We first qualitatively described the strategies using examples, and then gave a detailed description of the RL-based policy as a table-look-up between states and actions.

Of course, in this Chapter we have only tested the policies in interaction with a simulated environment. The ultimate evaluation is to test strategies in performance with real users. In the next Chapter we report on results from a detailed user study.

Part III
Evaluation and Application

Chapter 8
Comparing Reinforcement and Supervised Learning of Dialogue Policies with Real Users

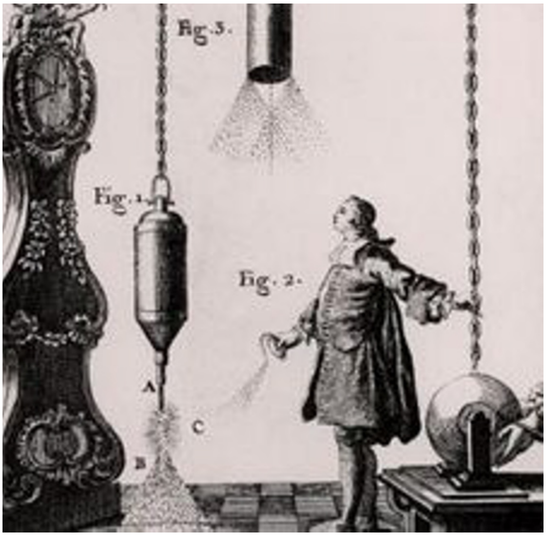

Fig. 8.1 Verification of a theory by real-world experiment.

In Chapter 7 we showed that Reinforcement Learning (RL) based strategies can significantly outperform supervised strategies, in interaction with a simulated environment. The ultimate test for dialogue strategies, however, is how they perform with real users. For real users it is often difficult to complete even relatively simple tasks using automated dialogue systems. They often get confused, frustrated, and irritated by unwieldy dialogue behaviour. These effects are hard to reproduce fully with simulated users. It is therefore necessary to test the learned strategies with real users to get subjective feedback, as well as collecting objective metrics.

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_8,

In this chapter we report results for testing the learned strategies with real users. We therefore port the policies into a working dialogue system, which we developed using the DUDE toolkit, as described in Section 8.1. In Section 8.2 we then explain the experimental setup for the user tests. The test results are analysed in Section 8.3, followed by a discussion in Section 8.4.

As a final step, we go on to evaluate our proposed development framework in Section 8.5. In particular, we compare aspects of simulated and real interactions in Section 8.5.1 and we evaluate different aspects of the reward function in Section 8.5.2).

8.1 Policy Integration into a Dialogue System

This book 'bootstraps' a simulated dialogue learning environment from WOZ data, which enables the design of optimal strategies before a working prototype is available. In order to test the validity of the overall framework, we need to show that the strategies obtained in such an environment are indeed suitable for dialogue interaction with real users. We therefore design a dialogue system where we can integrate and test the learned policies. The created system is an interactive interface to search in a large database of digital music files as described below.

8.1.1 The DUDE *Rapid Dialogue Development Tools*

Building a full-working dialogue system from scratch requires a substantial amount of manual labour. We therefore rely on the DUDE toolkit for system rapid development. DUDE is a Dialogue and Understanding Development Environment which facilitates rapid development of slot-based dialogue systems for the English language (Lemon and Liu, 2006).

8.1.1.1 The Development Process in DUDE

In order to develop a system with DUDE, the dialogue developer has to provide a mySQLdatabase and has to specify a *Process Model* (PM), see Figure 8.2. A PM describes the task as a hierarchical automaton with sub-tasks and states which are linked by conditional transitions (similar to Task Hierarchical Diagrams, as described in Section 2.2.1). This specification can be done via a GUI, or by providing a formatted input file. The DUDE toolkit then automatically generates a full working dialogue system for the given task, including context-sensitive ASR, a grammar for Natural Language Understanding, and a set of basic dialogue building blocks, such as sub-strategies and sub-routines.

One major advantage of the DUDE toolkit over other graphical toolkits, such as the CSLU toolkit,[1] is that DUDE provides a a simple interface to the Information State Update (ISU) approach to dialogue strategy design. The ISU approach allows developers to represent and track a rich representation of the dialogue context, which allows highly context-sensitive and sophisticated strategies to be specified. This dialogue context is managed by a library of generic routines and sub-strategies which come with the tool.

In addition to this task independent functionality, DUDE automatically derives a set of task-specific rules, according to the specified process model. The system developer then can edit the task-specific system prompts, can specify conditions for sub-task traversal, and s/he also may add some trigger phrases for the grammar. Other than that, the grammar and the speech recognition language model are automatically compiled from the provided database. In addition, a core grammar and lexicon also provide some additional generic functionality for slot-based systems.

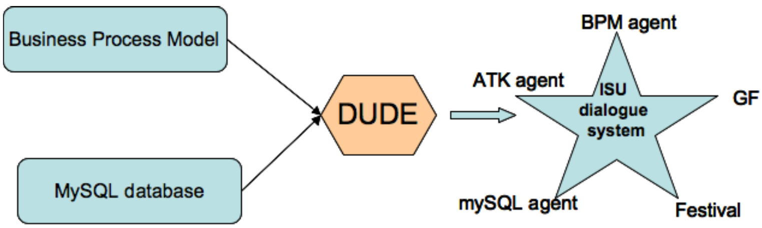

Fig. 8.2 The development process with the DUDE toolkit (graphic after Lemon and Liu (2006))

8.1.1.2 DUDE System Architecture and Generic Dialogue Management

The system architecture of DUDE is based on the Open Agent Architecture (OAA) (Cheyer and Martin, 2001), where several (mostly) JAVA-based agents interact with each other (also see Figure 8.2):

- A database agent for querying the MySQL database
- An ASR agent implementing the HTK recogniser (Young, 1995) using ATK with open speech recognition using n-grams (or alternatively Nuance Dragon[2] can also be used)
- A Process Model agent which serves as a default task planner
- A TTS agent for the Festival2 speech synthesiser (Taylor et al, 1998) (or alternatively CereProc[3] speech synthesiser can also be used)

[1] http://cslu.cse.ogi.edu/toolkit/ (24. April 2008); for a comprehensive introduction to the CSLU toolkit see (McTear, 2004).

[2] http://www.nuance.co.uk/naturallyspeaking/ (4. January 2011)

[3] http://www.cereproc.com/ (4. January 2011)

- A Grammatical Framework (GF) parser agent (Ranta, 2001)
- DIPPER for ISU dialogue management (Bos et al, 2002)
- An agent for text-based interaction (text in/out agent)
- An agent logging all information states (IS logger agent) following the format specified by Georgila et al (2005b)
- A GUI agent for multimodal dialogue systems (for example see (Lemon et al, 2006b)).

In addition, DUDE comes with a set of subroutines which cover the following generic core dialogue behaviour:

- mixed initiative and over-answering
- open-initiative dialogue start ("How may I help you?")
- explicit and implicit confirmation
- generic help, restart/start-over, repeat, and quit commands
- task switching (for applications with multiple tasks)
- database query and result presentation
- template-based Natural Language Generation (NLG)
- clarification of ambiguous semantic content (e.g. "Kid A. Do you mean album or song title?")
- possibility to leave a slot under-specified (for example the user might say "I don't mind", "I don't care", etc.)

8.1.2 *Extensions to* DUDE

In addition to the automatically generated behaviour, we added the following extensions to the dialogue functionality of DUDE. First of all, we changed the overall structure of the dialogue: from form-filling to information-seeking dialogue strategies (as defined in Section 3.4.1). Form filling strategies gather values for all slots before the database is queried. The retrieved items are only presented at the end of the dialogue (see Table 8.1 top). The disadvantage of such a procedure is that the dialogues can become unnecessarily long and unwieldy. Imagine the case where the user fills all the slots just to find out that the requested item is not in the database. Or the case where, there is only one possible candidate in the database after filling the first slot, then filling all the other slots should become redundant. For information-seeking dialogue strategies, in contrast, the database is queried every time the user provides a new constraint (for us this is when a new slot gets filled). The presentation phase can be entered at any point in the dialogue (see Table 8.1, bottom). Thus, one major decision for information-seeking dialogue systems is when to stop gathering new constraints (this is what we call "information acquisition") and present the retrieved items to the user.

The information acquisition phase includes the system acts for explicit (`explConf`) and implicit confirming slots (`implConfAskASlot`), asking for new slot values (`askASlot`), as well as the decision to enter the presentation phase (`presentInfo`);

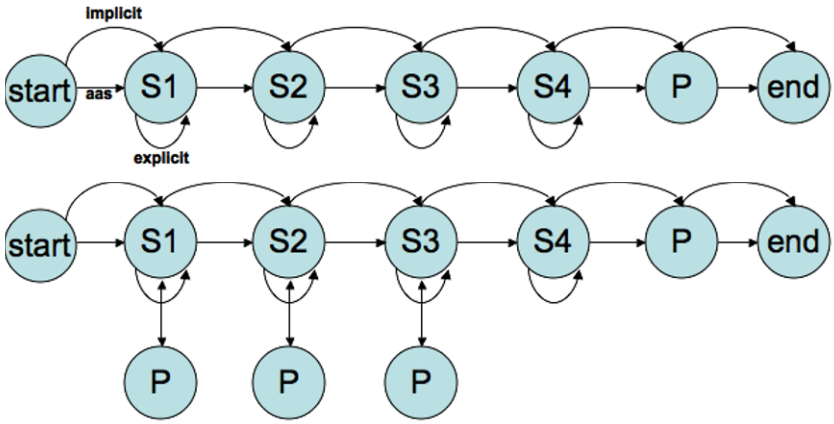

Table 8.1 Form filling (top) and information-seeking (bottom) dialogue strategies for a 4 slot problem ($S1-S4$) with different opportunities for entering the presentation phase (P). The arcs denote slot transitions with implicit/ explicit confirmation or by asking a new slot value (denoted as 'aas' in the figure)

cf. Figure 7.5 in Section 7.5.3. The confirmation strategies and the functionality for asking new slots values are automatically generated by standard generic routines in DUDE. In order to implement the additional presentation skills `presentInfoVerbal` and `presentInfoMM`, we had to extend the current functionality of DUDE as follows.

For verbal presentation we use a contrastive summary-based approach, following Polifroni and Walker (2006); Polifroni et al (2003). The retrieved items are presented by listing all slot features which overlap, and asking the user to provide a value for the slot which helps to distinguish the retrieved items (see Example 8.1.1). If there is more than one distinctive feature, the one with the highest information gain (entropy) with respect to the database content is chosen.

Example 8.1.1

User: *"Find me a song called Morning Bell."*

System: *"There are 3 results matching your query all by the artist Radiohead, all from the genre Alternative, and all having the song title Morning Bell, but a different album. You have the following options: 'Kid A' and 'I Might Be Wrong' and 'Amnesiac'. Which album?"*

For multimodal presentation we let the system verbally confirm the last slot value, and, at the same time, display a list of candidates on the screen (see Figure 8.3) where the user can select an item by clicking. Examples can be found in the dialogue in Appendix A.3.

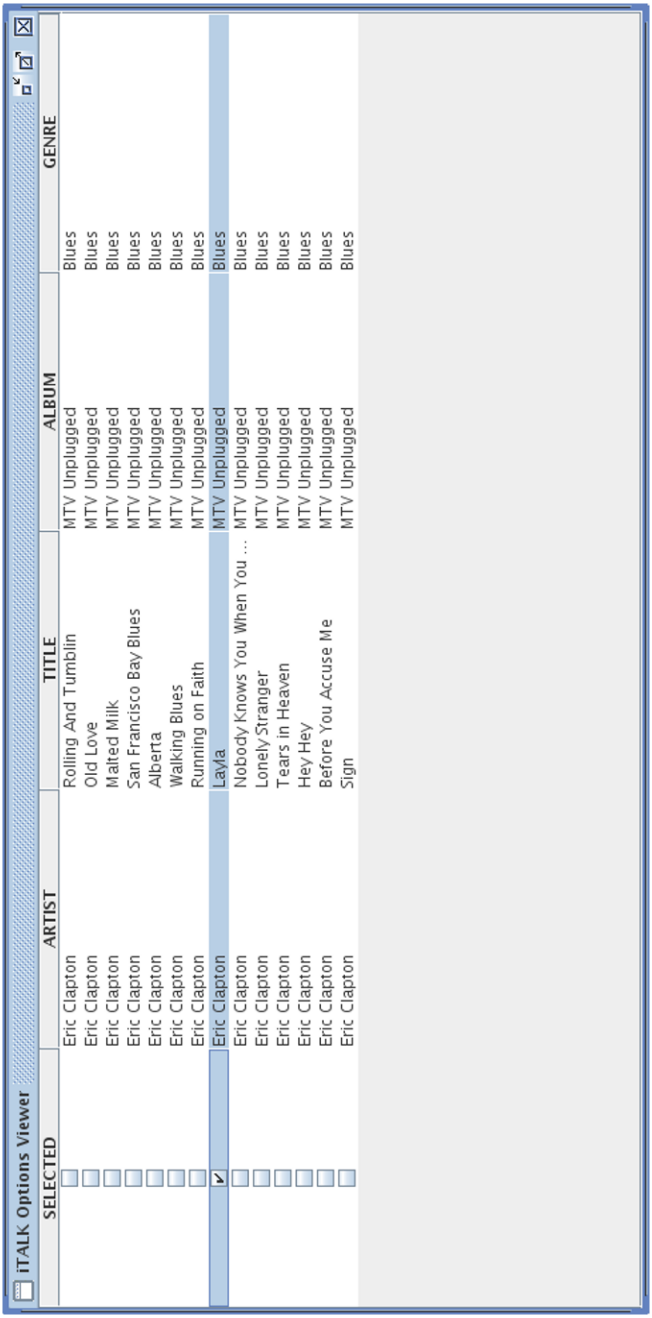

Fig. 8.3 List of retrieved items displayed in the *iTalk* system

The dialogue decisions for information acquisition and presentation are determined by the SL and RL policies as presented in the previous Chapter. All the other (generic) sub-strategies, such as greeting, closing, semantic clarification, or user commands like help, repeat, start-over, and so on, are implemented by generic DUDE routines. The SL policy follows a simple set of learned rules (see Algorithm 2 in Section 7.6.2). The RL policy is implemented using a table-look-up technique (see Section 7.11.4 and Appendix B). We also implemented the same constraints as used for learning (see Section 7.5.4).

We also handle the case when no results (zero items) are retrieved in the database. For simulation-based strategy learning we assumed that the user's goal is always contained in the database (see Section 7.8.7.2). The tasks for the user tests, however, are designed in such a way that not all the slot values are specified in advance (as further discussed in Section 8.2.4). We asked the user not to guess any additional values, but rather to ask for help or leave the option open (i.e. leaving a slot underspecified) in order to resemble a concrete 'search' scenario rather than a 'browsing' task. However, some subjects ignored this instruction and made a guess, which (occasionally) resulted in zero results. In this case, the system apologises and the dialogue restarts.

In addition, we added a context-sensitive help function. The existing help function only gives general information about the interaction with the system . The context-sensitive help function, in contrast, lists a number of examples of possible candidates for the current slot value. The system presents maximally 5 help items (see Example 8.1.2). Note that this threshold is manually set. Example 8.1.2 also illustrates the difference between the option to leave a slot unspecified ("I don't care") and the help function ("I don't know"). Context-sensitive help is always given if the user explicitly asks for help. The generic help only follows a time-out.

Example 8.1.2

User: *"I would like to hear a Blues song by Bob Dylan"*

System: *"Blues for type of music; searching for music by 'Bob Dylan. Which album?"*

User: *"Don't mind"*

System: *"Okay, it doesn't matter. What is the title of the song?"*

User: *"I don't know."*

System: *"You have several options. For example: 'Positively 4th Street' , 'I Dont Believe You', 'Most Of The Time', 'Black Diamond Bay' , and 'Desolation Row'. Which one would you like? '"*

Furthermore, we extended the functionality for Natural Language Understanding and database query. First, the system can now handle partial parses. For example, the user can say "Hendrix" instead of "Jimi Hendrix". Furthermore, the database is queried using "fuzzy" search (or partial string matching). For example, if the system only recognises the first two words of 'Subterranean Homesick Alien' by Radiohead, the song 'Subterranean Homesick Blues' by Bob Dylan is also returned (see also Chapter 7.2). The new functionality was implemented using Java-based OAA agents.

8.2 Experimental Setup

8.2.1 Technical Setup

The experimental conditions are similar to the WOZ study (see Chapter 6.1), i.e. we ask the users to solve similar tasks, and use similar questionnaires. With two exceptions: The WOZ study was performed in German, whereas the user test are performed in English. Therefore, a different database had to be used and tasks sets and user questionnaires had to be translated.[4] Furthermore, we decided to use typed user input rather than ASR (similar to chat interfaces used by other dialogue studies, e.g. (Bertomeu et al, 2006; Purver et al, 2003b)). The use of text input allows us to target the experiments to the dialogue management decisions on presentation strategies, and prevents ASR quality from interfering with the experimental results, especially since subjective user scores are highly sensitive to ASR noise (Hajdinjak and Mihelic, 2006). Both RL and SL wizard policies are trained to handle noisy conditions, so that they usually confirm user input, which makes dialogues longer but more reliable. The lack of noise in this experiment means that confirmation happens more than is strictly required (although there are still text input spelling mistakes), but the information presentation decisions are not affected.

8.2.2 Primary Driving Task

In contrast to the WOZ experiments we do not ask the experimental subjects to operate a driving simulator, but let them watch videos of a simulated driving task. In order to measure the cognitive load we give our experimental subjects a distraction task. We ask them to count intersections with traffic lights as they occur in these videos, and note down the number at the end of the dialogue. We can also use this measure to directly determine the cognitive load of the interaction. In particular, we use two different sets of driving videos.

- Driving set $D1$ consist of two videos, 'Simulated urban driving' (03 : 26)[5] and 'DriveSafety - Winter Driving Simulation' (04 : 07).[6]
- Driving set $D2$ only contains the video 'DriveSafety HyperDrive demo' (08 : 22).[7]

---

[4] Previous results indicate that human linguistic behaviour on the speech act level is very similar for English and German – at least for task-oriented dialogues in restricted domains (Rieser and Moore, 2005).

[5] `www.youtube.com/watch?v=rACwcqRKRg8` (1. September 2011)

[6] `www.youtube.com/watch?v=nLgQEYWkvoI` (1. September 2011)

[7] `www.youtube.com/watch?v=gGTDaHgO8zs` (41. September 2011)

Both sets are played back without sound, and the videos are played in a loop (in case the video ended before the user finished the task). Driving set $D1$ contains (3+3) intersections with traffic lights, set $D2$ contains 4.

The overall experimental setup is shown in Figure 8.4. The experimental subject is interacting with the dialogue system using a laptop with an external keyboard and mouse. The driving simulation video is projected on the wall behind. This setup allows the user to pay attention to both screens.

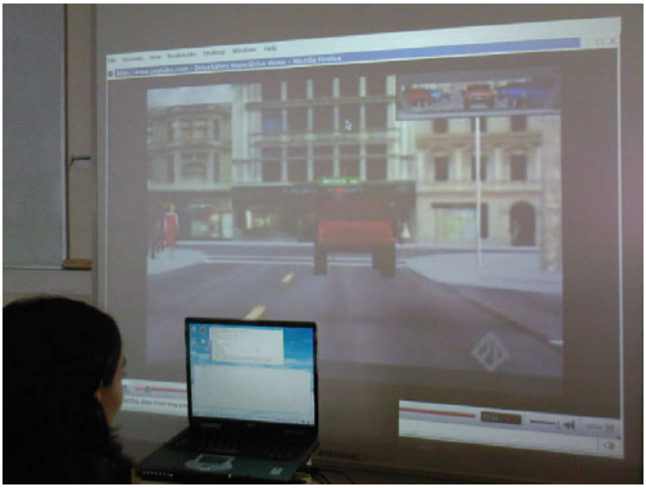

Fig. 8.4 Experimental setup – experimental subject interacting with the system (laptop) while watching a driving simulation video (wall projection)

8.2.3 Subjects and Procedure

We recruited 17 subjects (8 female, 9 male) for the experiment. All subjects were between 20 and 40 years old, and all but 3 subjects were able to touch type. The majority of our subjects were students (76.5%) and mainly non-native speakers of English (58.8%).

The experiment proceeded as follows. First, the experimental subject signs a statement of consent.Then s/he is given a sheet with experimental instructions to read. The instructions consist of a short description of the system's functionality, some explanation on possible commands (e.g. how to get help), and advice on how to proceed with the tasks. The "driving" task is to count traffic lights (as explained above). The tasks to solve with the system are simple search tasks where the subject is asked to search for a specific item (see next Section). Once the experimental subject has finished a task, the driving simulation video stops, and the s/he fills out a questionnaire (see Section 8.2.5). S/he then resumes with the next task and the driving video continues. After the subject has finished the first task set, the experiment leader switches between SL and RL dialogue policy, and the experimental subject starts with the next set of tasks (as described in the next Section). At the very end, the experimental subject may leave some comments. At the end of the experiment each participant received a small compensation of £5.

8.2.4 Task Types

Every experimental subject has to solve 12 tasks in total, 6 tasks are solved using the system executing the SL policy, and 6 tasks with the RL policy. This setup is called a "repeated-measures within-subject design", i.e. comparing strategy performance by subject and task. We therefore design 2 task sets (A and B) of 6 tasks each. These task sets are counter-balanced with strategy type (SL, RL) and driving simulation video ($D1$, $D2$) in a latin-square design (see (Field and Hole, 2003)).

The tasks are of 2 different types. In tasks with type 1 the subject is given a partial description of the item s/he should be looking for (e.g. genre, artist name, and/or album name). A task is successful if the user chooses a final (unique) song title matching the task. Task specifications of type 1 are of varying detail:

1a: Only one or two pieces of information are given, resulting in many possible candidates (> 5).
1b: The given information specifies a path leading to a small set of items to choose from (about 5 items).

A task of type 2 tells the user the specific song title s/he should be looking for. This song title has different degrees of ambiguity:

2a: Only one song with that title can be retrieved.
2b: The database contains 2-4 songs with that title.

Both task sets contain tasks of type 1 and 2 in the following order: $1a, 2b, 1b, 2b, 1a, 2a$.

8.2.5 User Questionnaires

The user fills out a questionnaire at the end of each task. The questions are of two different types. One part of the questionnaire targets task completion, the other part targets user satisfaction following Walker et al (2000). The first set of questions measure actual and perceived task completion. For measuring the actual task completion the user fills in the missing information in an attribute value matrix (AVM). For example, if the task scenario is described as *"You want to listen to a* **Blues** *song by the artist* **Bob Dylan**.", then the user has to fill in album and song title (as in Table 8.2).

genre	Blues
artist:	Bob Dylan
album:	
song title:	

Table 8.2 Example AVM for measuring task completion

The perceived task completion is measured as a binary variable, asking the user whether s/he completed the task in a yes-no question.

The other set of questions targets different dimensions of user satisfaction, using a subset of the questions asked in PARADISE (Walker et al, 2000). A discussion of the PARADISE questionnaire can be found in Section 2.2.2. In total the user has to rank the following 9 statements on a 7-point Likert Scale from 'totally disagree' to 'totally agree'. The corresponding dimensions of user satisfaction are given in brackets. In addition to PARADISE we also added questions targeting presentation timing (question 5) and presentation mode (questions 6,7).

1. In this conversation, it was easy to find what I was searching for. (**Task Ease**)
2. The system worked the way I expected it to, in this conversation. (**Expected Behaviour**)
3. In this task, I thought the system had no problems understanding me. (**NLU Performance**)
4. In this task, the system was easy to understand. (**TTS Performance**)
5. In this task, I thought the system chose to present the search results at the right time. (**Presentation Timing**)
6. In this task, I thought the number of items displayed on the screen was right. (**MM Presentation**)
7. In this task, I thought the amount of information presented in each spoken output was right. (**Verbal Presentation**)
8. In this task, I found that searching for music distracted me from the driving simulation. (**Cognitive Load**)
9. Based on my experience in this conversation, I would like to use this system regularly. (**Future Use**)

8.3 Results

In total, 204 dialogues with 1,115 turns[8] were gathered in this setup.

We now compare the results for the supervised wizard strategy and the RL strategy (as trained in the previous Chapter) in order to detect statistical differences in strategy performance. In particular, we compare subjective user ratings (as obtained

[8] A turn is defined as in (Paek and Chickering, 2005): it begins at the start of each new user action (since all system actions respond to user actions). A user action can be a verbal/typed utterance, or a click (i.e. the user selected an item on the screen).

from the questionnaires) and objective measures of dialogue performance (such as dialogue length).

8.3.1 Subjective User Ratings

We use the Wilcoxon signed-rank test for significance testing between the user ratings. The Wilcoxon signed-rank test is the non-parametric alternative to the paired t-test.[9] In the following we first report on the results for different measures of User Satisfaction, then we report on task related user ratings.

8.3.1.1 User Ratings on User Satisfaction

Measure	**Wizard/SL Strategy**	**RL Strategy**
Task Ease	4.78($\pm$1.84)	5.51($\pm$1.44)***
timing	4.42 ($\pm$1.84)	5.36 ($\pm$1.46)***
MM Presentation	4.57 ($\pm$1.87)	5.32 ($\pm$1.62)***
Verbal Presentation	4.94 ($\pm$1.52)	5.55 ($\pm$1.38)***
Future Use	3.86 ($\pm$1.44)	4.68 ($\pm$1.39)***

Table 8.3 Comparison of mean user ratings for SL wizard baseline and RL policies(with standard deviation $\pm$); *** denotes statistical significance at $p < .001$

The results for user ratings on different dimensions of User Satisfaction are displayed in Table 8.3. All the observed differences have a medium effects size ($r \geq |.3|$) (see (Cohen, 1992)). For all dimensions from the questionnaire, except for Cognitive Load, the RL-based policy performs significantly ($p < .001$) better than the SL-based policy. The total gain of the RL over the SL policy is on average 0.7 points on a 7-point Likert Scale (10%). The relative improvement for Task Ease (i.e. the measure optimised by the reward function, see Chapter 7.9) is 15.3%; for Future Use 23.52%.

An error analysis showed that in 41.2% of the cases the SL policy followed the same strategy as the RL policy. This is mainly due to the fact that the SL policy applies a frequency-based decision for starting the information presentation phase (with $P = 0.48$, see Section 7.6.2). Nevertheless, the cases where the RL-based system does behave differently (i.e. 58.8% of the time) cause a significant increase for

9 Although most previous work uses a paired t-test to analyse user questionnaires, this method assumes parametric data which is problematic for data on an ordinal scale, as typically obtained from user questionnaires. We also run a t-test were the results from the non-parametric test are confirmed. For non-parametric data, the results for the non-parametric tests are said to be more reliable (Field, 2005). We therefore report on those.

most of subjective ratings, indicating that the distinctive cases have a strong impact on user judgement.

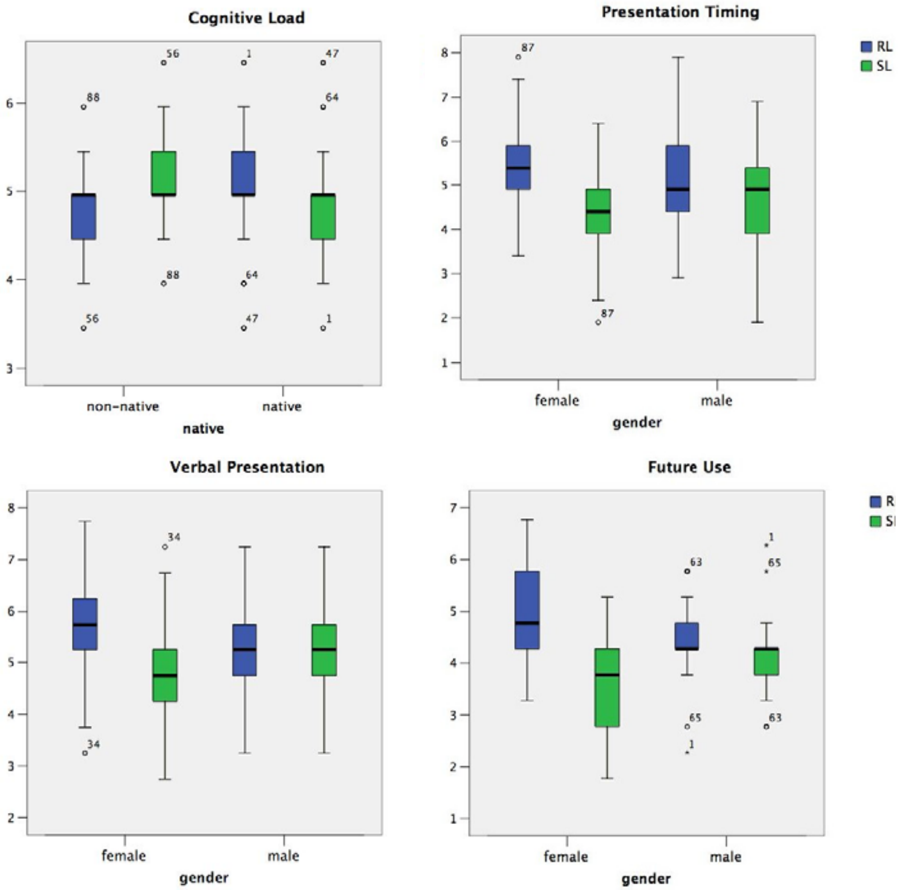

Table 8.4 Box plots (with outliers) illustrating the effects of gender and native vs. non-native speakers on user perception

Furthermore, we find that gender, as well as native versus non-native speaker of English, have an effect on user ratings. We find an effect for native versus non-native speakers for the Cognitive Load: the RL-based strategy distracts non-native speakers significantly less from the driving task ($z = -2.67; p = .008$) whereas for native speakers the reverse seems to be true (although differences are not significant here), see Table 8.4, top left.

It also seems that gender does influence how the presentation strategies are evaluated: male participants rate rather indifferently between the RL and the wizard strategy for most of the questions, whereas female participants clearly favour the RL-based strategy in terms of Presentation Timing, Multimodal and Verbal Presentation, and Future Use (see Table 8.4, top right and bottom row). Similar gender effects are reported by other studies on multimodal output presentation, e.g. (Foster and Oberlander, 2006; Jokinen and Hurtig, 2006; Robertson et al, 2004). Note, that for the WOZ study we do not find any gender differences.[10]

8.3.1.2 Comparison of Judgements Related to Task Success

We also asked the user to fill in questions related to task success (see Section 8.2.5). The actual task completion is measured according to an AVM, where filling all the requested slots corresponds to 100%. For example, if 2 slots are left to fill in (and two are specified by the task description), and the user just provides one of them, then task completion is evaluated as 50%. For actual task completion almost the maximal score is reached: RL= 93.7% SL= 90.0%. There are no significant differences between the policies.

The perceived task success is measured as a binary variable.[11] Here almost the maximum score is reached since for RL ($taskCompletion_{RL} = .98$) and for SL ($taskCompletion_{SL} = .96$). Again, there are no significant differences between the policies. This indicates that the system was very usable in general.

For the secondary task, counting traffic lights, no significant difference in performance is found (mean counting errors for RL= 2.0 and for SL= 2.3). However, for women there is a tendency to make fewer errors when using the RL-based dialogue strategy (mean counting errors for RL= 2.3 and for SL= 3.0, p=.084). Interestingly enough, there was no correlation between how much people felt distracted from the driving simulation (Cognitive Load) and how many errors they made in counting traffic lights.

[10] A related question is, whether the observed differences are due to gender specific preferences, or if they are due to differences in how men and women use subjective ranking scales (e.g. Likert Scales). For example, when using bipolar ranking scales respondents may avoid using extreme response categories ("central tendency bias"); or tend to agree with statements as presented ("acquiescence bias"); or try to portray themselves in a more favourable light ("social desirability bias"), (see `http://en.wikipedia.org/wiki/Likert_scale` (11. September 2011) for a discussion of the ratings produced by Likert Scales). In our study, men used the undecided middle value more often than women.

[11] Note that for the binary measure of perceived task success McNemar's test was used instead of the Wilcoxon signed-rank test, because the latter assumes ranked order. McNemar's test is a non-parametric method used on nominal data to determine whether the row and column marginal frequencies are equal, similar to the χ^2 test, but for binary data (Field, 2005).

8.3.1.3 Comments by the Users

At the end of the experiment experimental subjects were encouraged to leave their own comments. 8 out of 17 users (47.06%) left a comment behind. 5 out of 8 users criticised the insufficient TTS quality, especially for proper names. Some report that they were not able to understand some of the names of presented items. 2 users would prefer a speech-based interface, rather than typing. 2 users asked for more multimodal output and earlier presentation of candidates by the system. (This is what the SL policy does "wrong".) One user comments that it felt unnatural to choose songs by genre. Some others participants also noted that they had difficulties with specifying genre. Similar problems about choosing songs by genre are reported by Schulz and Donker (2006); Wang et al (2005).

We believe that these comments are very valuable, as they illustrate some major bottlenecks in strategy evaluation. The experimental design for evaluating dialogue system is complex and difficult to control, as there are many variables which can influence the perception of the system and its strategies. For example, the verbal presentation strategies suffered from the bad TTS quality, as well as task success. As such, one might consider learning generation strategies which take TTS quality into account (Boidin et al, 2009).

8.3.2 Objective Dialogue Performance

We now analyse the system log-files and compare objective measures for dialogue performance. In Section 7.11.2 we have shown that RL-based policies significantly outperform supervised policies when interacting with a simulated environment. We now test whether we can replicate these results with real users.

We use a paired t-test[12] (with pair-wise exclusion of missing values) for comparing the performance of RL versus the supervised wizard strategy. The results are shown in Table 8.5. The dialogues of the RL strategy are significantly shorter ($p < .005$), while fewer items are displayed ($p < .001$), and the help function is used significantly less ($p < .003$). The mean performance measures for testing with real users are shown in Table 8.5. Also see example dialogues in Appendix A.3.

The dialogue performance is summarised in the reward function (see Section 7.9.4). The RL-based policy gains on average almost 18-times more reward, which is significantly ($p < .001$) higher than for the SL policy (see Table 8.5).

Finally, we compare the performance of different user groups using an ANOVA with planned comparison. We find no gender effects for any of the dialogue features. However, we do find an effect for native versus non-native speakers of English. Non-native speakers performed significantly ($F(89) = 24.3, p < .001$) better in the secondary driving task (average errors per task set: 0.3 ± 2.3) than native speakers (average errors: 3.0 ± 2.7) when using the RL-based strategy. For the SL-based

[12] Here, a t-test is a valid option, as objective dialogue quality measures are on an interval scale and follow an approximately normal distribution.

Measure	**Wizard Strategy** mean(std)	**RL Strategy** mean(std)	paired t-test
turns	5.86 ($\pm$3.26)	5.07($\pm$2.94)*	t(101)=-1.9; p=.05 ;r=.19
GUI items	52.2 ($\pm$68.5)	8.73($\pm$4.34)**	t(25)=-3.3; p=.003; r=.55
help	.69 ($\pm$1.1)	0.34($\pm$.68)**	t(101)=-2.9; p=.005; r=.27
reward	-628.16($\pm$1808.3)	37.62($\pm$60.7)***	t(101)=3.7; p=.001; r=.35

Table 8.5 Direct comparison of objective measures for the WOZ and the RL-based policy; where * denotes statistical significance at $p < .05$, ** at $p < .005$, and *** at $p <.001$ with t-value and effect size r

policy native and non-native speakers performed about the same (error non-native: 1.4 ± 2.4; errors native: 1.5 ± 2.3). This confirms the results of the subjective user ratings for Cognitive Load (see Section 8.3.1.1): the RL-based policy helps non-native speakers to concentrate more on the secondary driving task. This suggests that the RL-based strategy is even more helpful for non-native speakers, as non-native speakers in general need to spend more effort on communicating in a foreign language.

8.4 Discussion of Real User Evaluation Results

In this Chapter we have shown that a dialogue strategy learned by Reinforcement Learning methods outperforms a supervised strategy when tested with real users. The RL-based policy gains on average almost 18-times more reward than the SL policy. The human users also subjectively rate the RL-based policy on average 10% higher.

In these user tests we decided to use typed user input rather than ASR. The use of text input allows us to target the experiments to the dialogue management decisions, and block ASR quality from interfering with the experimental results (Hajdinjak and Mihelic, 2006). However, ideally one would want to employ high-quality ASR to facilitate spoken interaction, as the user is in a eyes-busy situation (i.e. s/he is visually distracted by the driving simulation). Nevertheless, we are still able to make claims about the *relative* improvement of RL over the supervised wizard policy, as both were tested under the same conditions.

However, these user tests are still set up under laboratory conditions. Work by Ai et al (2007a) shows that there are some major differences between laboratory dialogues and the interaction between real users and a deployed system. For example, recruited subjects have no real goals that they want to achieve. For these reasons recent work focuses on evaluations with real users “in the wild” – that is , real users who are outside the laboratory and who have their own genuine information goals, rather than being supplied with a set of experimental tasks. See Black et al (2011) for examples of this type of evaluation.

In our case, we learn a strategy from data which was gathered in a controlled experiment, and this strategy is also tested under similar conditions. It is not clear how the learned strategy would perform while the user is driving a car (as for example tested by Mutschler et al (2007)).

In the next Section we take a first step towards investigating how well a learned strategy transfers between different operating conditions. We evaluate how the results transfer between the initial WOZ study, the simulated environment, and the tests with real (though still in the laboratory) users.

8.5 Meta-Evaluation

Here we introduce a final phase where we meta-evaluate the whole framework. This final step is necessary since WOZ experiments only *simulate* HCI. We therefore need to show that a strategy bootstrapped from WOZ data indeed transfers to real HCI. We first show that the results for simulated and real interaction are compatible (Section 8.5.1). We also meta-evaluate the reward function, showing that it is a stable, accurate estimate for real users' preferences (Section 8.5.2).

8.5.1 Transfer Between Simulated and Real Environments

We first test whether the results obtained in simulation transfer to tests with real users, following Lemon et al (2006a). We evaluate the quality of the simulated learning environment by directly comparing the dialogue performance measures between simulated and real interaction. This comparison enables us to make claims regarding whether a policy which is "bootstrapped" from WOZ data is transferable to real HCI. We first evaluate whether objective dialogue measures are transferable, using a paired t-test, comparing overall mean performance.

For the RL policy there is no statistical difference in overall performance (reward), dialogue length (turns), and the number of presented items (verbal and multimodal items) between simulated and real interaction (see Figure 8.5). This fact (that the performances are not different) indicates that the learned strategy transfers well to real settings. For the SL wizard policy the dialogue length for real users is significantly ($t(101) = 5.5, p < .001, r = .48$) shorter than in simulation. We conclude from an error analysis that this length difference is mainly due to the fact that real users tend to provide the most "informative" slot value (i.e. the most specific value from the experimental task description) right at the beginning of the task (and therefore more efficiently contribute to solve the task), whereas simulated users use a default ordering of slot values and most of the time they provide the slot value that the system was asking for (`provide_info`). This difference becomes more prominent for the SL wizard policy than for the RL-based policy, as the SL wizard policy in general asks more questions before presenting the information. In future work the user simulation therefore should learn optimal slot ordering, also see discussion in Section 7.11.5.2.

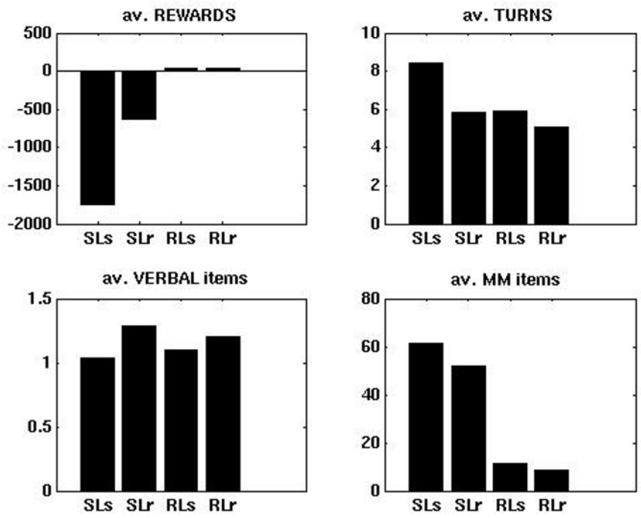

Fig. 8.5 Graph comparison of objective measures: SLs = SL policy in simulation; SLr = SL policy with real users; RLs = RL policy in simulation; RLr = RL policy with real users

8.5.2 Evaluation of the Learned Reward Function

We propose a new method for meta-evaluation of the reward (or "objective") function. One main advantage of RL-based dialogue strategy development is that the dialogue strategy can be automatically trained and evaluated using the same objective function (Walker, 2005). Despite its central aspect for RL, quality assurance for objective functions has received little attention so far. In fact, as noted in Section 7.9, the reward function is one of the most hand-coded aspects of RL (Paek, 2006).

Here, we bring together two strands of research for evaluating the reward function: one strand uses Reinforcement Learning to automatically optimise dialogue strategies, e.g. (Henderson et al, 2008; Rieser and Lemon, 2008b,c; Singh et al, 2002); the other focuses on automatic evaluation of dialogue strategies, e.g. the PARADISE framework (Walker et al, 1997), and meta-evaluation of dialogue metrics, e.g. (Engelbrecht and Möller, 2007; Paek, 2007). Clearly, automatic optimisation and evaluation of dialogue policies, as well as quality control of the objective function, are closely inter-related problems: how can we make sure that we optimise a system according to real users' preferences?

In Section 7.9 we constructed a data-driven objective function using the PARADISE framework, and used it for automatic dialogue strategy optimisation, following work by Walker et al (1998a). However, it is not clear how reliable such a predictive model is, i.e. whether it indeed estimates real user preferences. The models obtained with PARADISE often fit the data poorly (Engelbrecht and Möller, 2007). It is also not clear how general they are across different systems and user groups (Paek, 2007; Walker et al, 2000). Furthermore, it is not clear how they perform when being used for automatic strategy optimisation within the RL framework.

In the following we evaluate different aspects of the reward function. In subsection 8.5.2.1 we test the model stability in a test-retest comparison across different user populations and data sets. In subsection 8.5.2.2 we measure its prediction accuracy.

8.5.2.1 Reward Model Stability

We first test the reward model's stability by re-constructing it from the data gathered in the real user tests (see Chapter8) and comparing it to the original model constructed from the WOZ data. By replicating the regression model on different data sets we test whether the automatic estimate of Task Ease generalises beyond the conditions and assumptions of a particular experimental design. The resulting models are shown in Equations 8.1 to 8.3 , where $TaskEase_{WOZ}$ is the regression model obtained from the WOZ data,[13] $TaskEase_{SL}$ is obtained from the user test data running the supervised average wizard policy, and $TaskEase_{RL}$ is obtained from the user test data running the RL-based policy. They all reflect the same trends: longer dialogues (measured in turns) result in a lower Task Ease, whereas a good performance in the multimodal information presentation phase (multimodal score) will positively influence Task Ease. For the user tests almost all the tasks were completed; therefore task completion was only chosen to be a predictive factor for the WOZ model.

$$TaskEase_{WOZ} = 1.58 + .12 * taskCompl + .09 * mmScore - .20 * dialogueLength \tag{8.1}$$

$$TaskEase_{SL} = 3.50 + .54 * mmScore - .34 * dialogueLength \tag{8.2}$$

$$TaskEase_{RL} = 3.80 + .49 * mmScore - .36 * dialogueLength \tag{8.3}$$

To evaluate the obtained regression models we use two measures: how well they fit the data and how close the functions are to each other (model replicability). Both are measured using goodness-of-fit R^2. For the WOZ model the data fit was rather low ($R^2_{WOZ} = .123$),[14] whereas for the models obtained from the user tests the fit has improved ($R^2_{RL} = .48$, and $R^2_{SL} = .55$).

[13] In contrast to the model in Equation 7.7 we now include the constant in the regression.

[14] For R^2 we use the adjusted values.

Next, we compare how well the models from different data sets fit each other. While the models obtained from the user test data show almost perfect overlap ($R^2 = .98$), the WOZ model differs ($R^2 = .22$) in the sense that it assigns less weight to dialogue length and the multimodal presentation score, and more weight is assigned to task completion. Task completion did not play a role for the user tests, as mentioned above. This shows that multimodal presentation and dialogue length become even more important once the tasks are being completed. Overall, then, the data-driven reward model is relatively stable across the different data sets (WOZ, real users with the SL policy, and real users using the RL policy).

8.5.2.2 Reward Model Performance: Prediction Accuracy

We now investigate how well these reward models generalise by testing their prediction accuracy. Previous research evaluated two aspects: how well a given objective function / reward model is able to predict unseen scores from the original system (Engelbrecht and Möller, 2007), and how well it is able to predict unseen scores of a new/different system (Walker et al, 2000). We evaluate these two aspects as well, the only difference is that we use the Root Mean Standard Error (RMSE) instead of R^2 for measuring the model's prediction accuracy. The RMSE is a frequently-used measure of the differences between values predicted by a model or an estimator and the values actually observed. It is defined over $[0,\infty]$, where 0 indicates perfect overlap. The maximum RMSE possible (= worst case) in our setup is 7 for SL/RL and 5 for WOZ. In order to present results from different scales we also report the percentage of the RMSE of the maximum error (% error). RMSE is (we argue) more robust for small data sets.[15]

First, we measure the predictive power of our models within the same data set using 10-fold cross validation, and then across the different systems by testing models trained on one system to predict perceived Task Ease scores for another system, following a method introduced by Walker et al (2000).

The results for comparing the RMSE for training and testing within data sets (ID 1-3) and across data sets (ID 4–5) are shown in Table 8.7. RMSE measures the average of the square of the "error". As such, lower RMSE values are better. The contrary is true for R^2, where "1" indicates perfect overlap between two functions.

[15] In particular, we argue that, by correcting for variance, R^2 can lead to artificially good results when using small tests sets (which typically vary more) and is sensitive to outliers (see Equation 8.4). RMSE instead measures the (root) mean difference between actual and predicted values (see Equation 8.5).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2}{\sum_{i=1}^{n}(y - \overline{y})} \tag{8.4}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2} \tag{8.5}$$

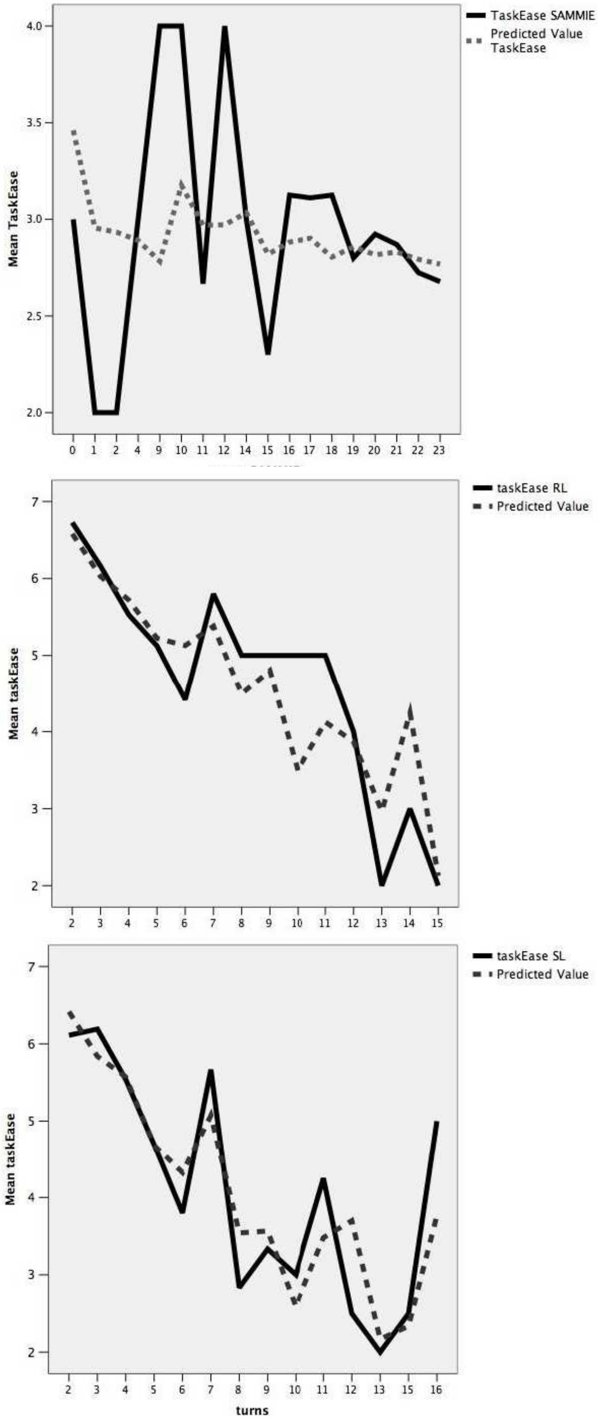

Table 8.6 Average Task Ease ratings for dialogues of different length (in turns); the solid lines are the true ratings and the dashed line the predicted values; from top: RL, SL wizard, WOZ data

ID	train	test	RMSE	% error
1	WOZ	WOZ	0.82	16.42
2	SL	SL	1.27	18.14
3	RL	RL	1.06	15.14
4	RL	SL	1.23	17.57
5	SL	RL	1.03	14.71

Table 8.7 Prediction accuracy for models within (1–3) and across (4–5) data sets

The results show that predictions according to PARADISE can lead to accurate test results despite the low data fit. While for the regression model obtained from the WOZ data the fit was 10-times lower than for SL/RL, the prediction performance is comparably good (see Table 8.7, ID 1–3). The models also generalise well across systems (see Table 8.7, ID 4–5).

Table 8.6 visualises the results (ID 1–3): mean values for predicted and for true ratings are plotted per turn, see Engelbrecht and Möller (2007). The top two graphs in the table show that the predicted mean values are fairly accurate for the SL and RL objective functions. The graph at the bottom indicates that the predictions are less accurate for the WOZ data, especially for low numbers of turns. This seems to contradict the previous results in Table 8.7, which show low error rates for the WOZ data. However, this is due to the fact that most of the observations in the WOZ data set are in the region were the predictions are accurate (i.e. most of the dialogues in the WOZ data are over 14 turns long, where the curves converge).

We conclude that, according to our measures, an objective function obtained from WOZ data is a valid first estimate of real users' preferences. Despite a low fit to the initial data, the objective function obtained from WOZ data makes accurate and useful predictions for automatic dialogue evaluation/reward. The models obtained from the tests with a real system follow the same trends, but can be seen as more reliable estimates of the objective function in this domain. In future work we recommend incrementally training a system according to improved representations of real user preferences, for example gathered online from a deployed spoken dialogue system.

8.6 Summary

This Chapter has presented the results of an evaluation of learned dialogue strategies with real users. The RL-based policy gains significantly more reward than the SL policy, with significantly shorter dialogues and fewer items being displayed to the users. We also presented an analysis of dialogue properties and the reward function across the different data sets (Wizard-of-Oz, simulated user, and real users).

What other types of problems can our overall methodology be applied to, and how successful could it be? We explore this issue in the next chapter, as we describe very recent work in extending this methodology to related problems in Natural Language Generation (NLG), again with a real-user evaluation.

Chapter 9
Adaptive Natural Language Generation

Fig. 9.1 Tools for generating language – one of the first typewriters, developed by Peter Mitterhofer in 1864, displayed in the Technisches Museum Wien

This Chapter shows how the framework developed throughout the book can be applied to a related set of problems, in Natural Language Generation (NLG) for interactive systems. It therefore provides some evidence for the generality of our approach, as well as drawing out some new insights regarding its application. This approach to DM and NLG has recently been explored by other researchers (Deth-

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_9,

lefs and Cuayahuitl, 2010; Dethlefs and Cuayáhuitl, 2011; Janarthanam and Lemon, 2010c), and a utility-based approach to NLG is also discussed by van Deemter (2009a).

Here we present a novel approach to generating Information Presentation (IP) strategies in Spoken Dialogue Systems using a data-driven statistical optimisation framework for content planning and attribute selection. Again, we collected data in a Wizard-of-Oz (WOZ) experiment, where we use the observed human behaviour as a baseline to measure the improvement of optimised policies, developed from this data using Reinforcement Learning methods. Firstly, in simulated environments, we will show that the optimised policies significantly outperform the human "wizard" baselines in a variety of generation scenarios: while the wizards were able to attain up to 87.6% of the possible reward on this task, the RL policies are significantly better in 5 out of 6 scenarios, gaining up to 91.5% of the maximum possible reward. The RL policies perform especially well in more complex scenarios, and we show that adding predictive "lower level" features (e.g. from a surface realiser and the predicted user reaction) is important for optimising IP strategies. This provides new insights into the nature of the IP problem for SDS.

Finally, we also report on an evaluation of the optimised IP strategy with real users of a tourist information system, compared to a state-of-the-art baseline, in Section 9.6. For further detail also see (Lemon et al, 2012; Liu et al, 2009; Rieser and Lemon, 2009a, 2010a; Rieser et al, 2010, 2011).

9.1 Introduction

Information Presentation (IP) is often a "grey area" when building Spoken Dialogue Systems (SDS); it falls somewhere in-between Dialogue Management (DM) and Natural Language Generation (NLG) and it often is not clear which of these components (if any) should take responsibility for it. A consequence of this is that in the great majority of deployed SDS the Information Presentation phase is carried out by hand-crafted rule-based templates rather than being developed using a data-driven methodology.

In the following we show that IP for SDS can be treated as a data-driven joint optimisation problem, and that this outperforms human 'wizards' on a particular IP task (presenting sets of search results to a user). To achieve this the IP module needs to take features from both DM and lower-level aspects of NLG into account.

9.1.1 Previous Work on Information Presentation in SDS

Evaluation of SDS suggests that the IP phase is the primary contributor to dialogue duration (Walker et al, 2001a), and as such, is a central aspect of SDS design. During this phase the system returns a set of items ("hits") from a database, which match the user's current search constraints. An inherent problem in this task is the trade-off between presenting "enough" information to the user (for example helping them to feel confident that they have a good overview of the search results) versus keeping the utterances short and understandable, especially when using spoken output.

In the information presentation problem there there are many decisions available for exploration. For instance, which presentation strategy to apply (*NLG strategy selection*), how many attributes of each item to present (*attribute selection*), how to rank the items and attributes according to different models of user preferences (*attribute ordering*), how many (specific) items to tell them about (*coverage*), how many sentences to use when doing so (*syntactic planning*), and which words to use (*lexical choice*) etc. All these parameters (and potentially many more) can be varied, and ideally, jointly optimised based on user judgements.

Prior work on **Content or Attribute Selection** has used a "Summarize and Refine" approach. This method employs utility-based attribute selection with respect to how each attribute (e.g. price or food type in restaurant search) of a set of items helps to narrow down the user's goal to a single item (Chung, 2004b; Polifroni and Walker, 2006, 2008). Related work explores a user modelling approach, where attributes are ranked according to user preferences (Demberg and Moore, 2006; Winterboer et al, 2007). Our data collection (see section 9.3) and training environment incorporate these approaches.

There are also a variety of **IP strategies** for structuring information (see examples in Table 9.1). For example, a SUMMARY policy is used in (Demberg and Moore, 2006; Polifroni and Walker, 2008), whereas a COMPARE strategy is used by Nakatsu (2008); Walker et al (2007). Most work in SDS uses a RECOMMEND policy, e.g. (Young et al, 2007).

In an early proof-of-concept study (Rieser and Lemon, 2009b) we showed that each of these strategies has its own strengths and drawbacks, dependent on the particular context in which information needs to be presented to a user. Here, we also explore possible *combinations* of the strategies, for example SUMMARY followed by RECOMMEND, e.g. (Whittaker et al, 2002), see Figure 9.2.

Our work has been the first to apply a data-driven method to this whole decision space, and to show the utility of both lower-level and higher-level features for this problem. Previous work focused on individual aspects of the problem (e.g. how many attributes to generate, or when to use a SUMMARY), using a pipeline model for SDS with DM features as input, and where NLG has no knowledge of lower level features (e.g. behaviour of the surface realiser). In the following we use our Reinforcement Learning framework to explore the contextual features for making these decisions, and propose a new joint optimisation method for IP strategies combining content structuring and attribute selection. We show that, as well as higher level DM features, optimal IP needs access to "forward looking" low-level features of surface realisation and a model of probable user reactions.

9.2 NLG as Planning Under Uncertainty

We follow the overall framework of NLG as planning under uncertainty (Lemon, 2008; Rieser and Lemon, 2009b), where each NLG action is a sequential decision point, based on the current dialogue context and the expected long-term utility or "reward" of the action. Other recent approaches describe this task as planning, e.g. (Koller and Petrick, 2008), or as contextual decision making according to a cost function (van Deemter, 2009b), but not as a statistical planning problem. Below, we apply this framework to Information Presentation strategies in SDS, where the example task is to present a set of search results (e.g. restaurants) to users. In particular, we consider 7 possible policies for structuring the content (see Figure 9.2): Recommending one single item, comparing two items, summarising all of them, or ordered combinations of those actions, e.g. first summarise all the retrieved items and then recommend one of them. The IP module has to decide which action to take next, which attributes to mention (cuisine, price range, location, food quality, and/or service quality), and when to to stop generating.

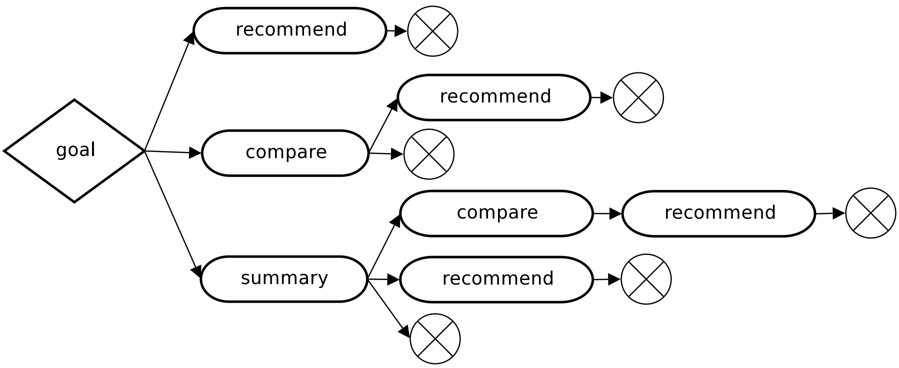

Fig. 9.2 Possible Information Presentation structures (X=stop generation)

9.3 Wizard-of-Oz Data Collection

In an initial Wizard-of-Oz (WOZ) study, we asked humans (our "wizards") to produce good IP actions in different dialogue contexts, when interacting in spoken dialogues with other humans (the users), who believed that they were talking to an automated SDS. The wizards were experienced researchers in SDS and were familiar with the search domain (restaurants in Edinburgh). They were instructed to select IP structures and attributes for NLG so as to most efficiently allow users to find a restaurant matching their search constraints.

Strategy	Example utterance
SUMMARY no UM	I found 26 restaurants, which have Indian cuisine. 11 of the restaurants are in the expensive price range. Furthermore, 10 of the restaurants are in the cheap price range and 5 of the restaurants are in the moderate price range.
SUMMARY UM	26 restaurants meet your query. There are 10 restaurants which serve Indian food and are in the cheap price range. There are also 16 others which are more expensive.
COMPARE by Item	The restaurant called Kebab Mahal is an Indian restaurant. It is in the cheap price range. And the restaurant called Saffrani, which is also an Indian restaurant, is in the moderate price range.
COMPARE by Attribute	The restaurant called Kebab Mahal and the restaurant called Saffrani are both Indian restaurants. However, Kebab Mahal is in the cheap price range while Saffrani is moderately priced.
RECOMMEND	The restaurant called Kebab Mahal has the best overall quality amongst the matching restaurants. It is an Indian restaurant, and it is in the cheap price range.

Table 9.1 Example realisations, generated when the user provided `cuisine=Indian`, and where the wizard has also selected the additional attribute `price` for presentation to the user

9.3.1 Experimental Setup and Data Collection

We collected 213 dialogues with 18 subjects and 2 wizards (Liu et al, 2009). Each user performed a total of 12 tasks, where no task set was seen twice by any one wizard. The majority of users were from a range of backgrounds in a higher education institute, in the age range 20-30, native speakers of English, and none had prior experience of Spoken Dialogue Systems. After each task the user answered a questionnaire on a 6 point Likert scale, regarding the perceived generation quality in that task. The wizards' IP strategies were highly ranked by the users on average (4.7), and users were able to select a restaurant in 98.6% of the cases. No significant difference between the wizards was observed.

The data contains 2236 utterances in total: 1465 wizard utterances and 771 user utterances. We automatically extracted 81 features (e.g #sentences, #DBhits, #turns, #ellipsis) from the XML logfiles after each dialogue.

The task for the wizards was to decide which IP structure to use next, which attributes to mention, and whether to stop generating, given varying numbers of database matches, varying prompt realisations, and varying user behaviour. Wizard utterances were synthesised using a state-of-the-art text-to-speech engine. The user speech input was delivered to the wizard using Voice Over IP.

9.3.2 Surface Realiser

In the Wizard-of-Oz environment we implemented a surface realiser for the chosen IP structures and attribute choices, in order to realise the wizards' choices in real time. This generator is based on data from the stochastic sentence planner SPaRKy (Stent et al, 2004b). We replicated the variation observed in SPaRKy by analysing high-ranking example outputs (given the highest possible score by the SPaRKy judges) and implemented the variance using dynamic sentence generation. The realisations vary in sentence aggregation, aggregation operators (e.g. ‘and’, period, or ellipsis), contrasts (e.g. ‘however’, ‘on the other hand’) and referring expressions (e.g. ‘it’, ‘this restaurant’) used. The length of an utterance also depends on the number of attributes chosen, i.e. the more attributes the longer the utterance. All of these variations were logged.

In particular, we realised the following IP actions (see examples in Table 9.1):

- SUMMARY of all matching restaurants with or without a User Model (UM), following
  (Polifroni and Walker, 2008). The approach using a UM assumes that the user has certain preferences (e.g. cheap) and only tells him about the relevant items, whereas the approach with no UM lists all the matching items.
- COMPARE the top 2 restaurants by Item or by Attribute.
- RECOMMEND the top-ranking restaurant (according to UM).

Note that there was no discernible pattern in the data about the wizards’ decisions between the UM/no UM and the byItem/byAttribute versions of the strategies. In this study we will therefore concentrate on the higher level decisions (SUMMARY VS. COMPARE VS. RECOMMEND) and model these different realisations as noise in the realiser.

9.3.3 Human “Wizard” Baseline Strategy

We analysed the WOZ data to explore the best-rated strategies (the top scoring 50%) that were employed by humans for this task. Here we used a variety of Supervised Learning methods to create a model of the highly rated wizard behaviour. The best performing method was Rule Induction (JRip) [1]. The model achieved an accuracy of 43.19% which is significantly ($p < .001$) better than the majority baseline (34.65%). The resulting rule set is shown in Figure 9.3.

The features selected by this model were only “high-level” features, i.e. the input (previous action, number of database hits) that an IP module receives as input from a Dialogue Manager. We further analysed the importance of different features using feature ranking and selection methods, finding that the human wizards in this specific setup did not pay significant attention to any lower level features, e.g. from surface realisation.

Nevertheless, note that the wizard policy achieves up to 87.6% of the possible reward on this task, as we show in section 9.5.2, and so can be considered a serious baseline against which to measure performance. Below, we will show that Reinforcement Learning (RL) produces a significant improvement over the strategies

[1] The WEKA implementation of Cohen (1995)’s RIPPER.

```
IF (dbHits <= 9)& (prevNLG = summary):
   THEN nlgStrategy=compare;
IF (dbHits = 1):
   THEN nlgStrategy= Recommend;
IF(prevNLG=summaryRecommend)&(dbHits>=10):
   THEN nlgStrategy= Recommend;
ELSE nlgStrategy=summary;
```

Fig. 9.3 Rules learned by JRip for the wizard model (‘`dbHits`’= number of database matches, ‘`prevNLG`’= previous NLG action)

present in the original data, especially in cases where RL has access to “low-level” features of the context.

9.4 The Simulation / Learning Environment

Here we “bootstrap” a simulated training environment from the WOZ data, following the method presented in earlier chapters.

9.4.1 User Simulations

User Simulations are commonly used to train strategies for Dialogue Management, see for example (Young et al, 2007). A user simulation for NLG is very similar, in that it is a predictive model of the most likely next user act.[2] However, this NLG predicted user act does not actually change the overall dialogue state (e.g. by filling slots) but it only changes the generator state. In other words, the NLG user simulation tells us what the user is most likely to do next, *if we were to stop generating now*.

We are most interested in the following user reactions:

1. `select`: the user chooses one of the presented items, e.g. *“Yes, I’ll take that one.”*. This reply type indicates that the information presentation was sufficient for the user to make a choice.
2. `addInfo`: The user provides more attributes, e.g. *“I want something cheap.”*. This reply type indicates that the user has more specific requests, which s/he wants to specify after being presented with the current information.
3. `requestMoreInfo`: The user asks for more information, e.g. *“Can you recommend me one?”, “What is the price range of the last item?”*. This reply type

[2] Similar to the internal user models applied in recent work on POMDP (Partially Observable Markov Decision Process) dialogue managers (Gasic et al, 2008; Henderson and Lemon, 2008; Young et al, 2007) for estimation of user act probabilities.

indicates that the system failed to present the information the user was looking for.

4. `askRepeat`: The user asks the system to repeat the same message again, e.g. *"Can you repeat?"*. This reply type indicates that the utterance was either too long or confusing for the user to remember, or the TTS quality was not good enough, or both.
5. `silence`: The user does not say anything. In this case it is up to the system to take initiative.
6. `hangup`: The user closes the interaction.

We build user simulations using n-gram models of system (s) and user (u) acts, as first introduced by Eckert et al (1997). In order to account for data sparsity, we apply different *discounting* ("smoothing") techniques including automatic *back-off*, using the CMU Statistical Language Modelling toolkit (Clarkson and Rosenfeld, 1997). We construct a **bi-gram** model[3] for the users' reactions to the system's IP structure decisions ($P(a_{u,t}|IP_{s,t})$), and a **tri-gram** (i.e. IP structure + attribute choice) model for predicting user reactions to the system's combined IP structure and attribute selection decisions: $P(a_{u,t}|IP_{s,t}, attributes_{s,t})$.

We evaluate the performance of these models by measuring dialogue similarity to the original data, based on the Kullback-Leibler (KL) divergence, as also used by, e.g. (Cuayáhuitl et al, 2005; Janarthanam and Lemon, 2009; Jung et al, 2009). We compare the raw probabilities as observed in the data with the probabilities generated by our n-gram models using different discounting techniques for each context, see Table 9.2. All the models have a small divergence from the original data, suggesting that they are reasonable simulations for training and testing NLG policies.

The absolute discounting method for the bi-gram model is most dissimilar to the data, as is the linear discounting method for the tri-gram model, i.e. the models using these discounting methods have the highest KL score. The best performing methods (i.e. most similar to the original data), are linear discounting for the bi-gram model and GoodTuring for the tri-gram. We use the most similar user models for system training, and the most dissimilar user models for testing NLG policies, in order to test whether the learned policies are robust and adaptive to unseen dialogue contexts.

Discounting method	bi-gram US	tri-gram US
WittenBell	0.086	0.053
GoodTuring	0.086	0.051
Absolute	0.091	0.075
Linear	0.011	0.090

Table 9.2 Kullback-Leibler divergence for the different User Simulations (US)

[3] Where $a_{u,t}$ is the predicted next user action at time t, $IP_{s,t}$ was the system's Information Presentation action at t, and $attributes_{s,t}$ is the attributes selected by the system at t.

9.4.2 Database Matches and "Focus of Attention"

An important task of Information Presentation is to support the user in choosing between all the available items (and ultimately in selecting the most suitable one) by structuring the current information returned from the database. We therefore model the user's "focus of attention" created by structuring information with different numbers of attributes. We implement this shift of the user's focus analogously to discovering the user's goal in Dialogue Management: every time the predicted next user act is to add information (`addInfo`), we infer that the user is therefore only interested in a subset of the previously presented results and so the system should only talk about this new subset of database items in the rest of the generated utterance. For example, the user's focus after the SUMMARY (with UM) in Table 9.1 is $DBhits = 10$, since the user is only interested in cheap, Indian places.

9.4.3 Data-driven Reward Function

The reward function is constructed from the WOZ data, using a stepwise linear regression, following the PARADISE framework (Walker et al, 2000). This model selects the features which significantly influenced the users' ratings for the NLG strategy in the WOZ questionnaire. We also assign a value to the user's reactions ($valueUserReaction$), similar to optimising task success for DM (Jung et al, 2009). This reflects the fact that good IP strategies should help the user to `select` an item ($valueUserReaction = +100$) or provide more constraints `addInfo` ($valueUserReaction = \pm 0$), but the user should not do anything else ($valueUserReaction = -100$). The regression in equation 9.1 ($R^2 = .26$) indicates that users like to be focused on a small set of database hits (where $\#DBhits$ ranges over [1-100]), which will enable them to choose an item ($valueUserReaction$), while keeping the IP utterances short (where $\#sentence$ is in the range [2-18]):

$$\begin{aligned} Reward = (-1.2)\#DBhits \\ +(.121)valueUserReaction \\ -(1.43)\#sentence \end{aligned} \tag{9.1}$$

Note that the worst possible reward for an NLG move is therefore $(-1.20 \times 100) - (.121 \times 100) - (18 \times 1.43) = -157.84$. This is achieved by presenting 100 items to the user in 18 sentences, in such a way that the user ends the conversation unsuccessfully. The top possible reward is achieved in the rare cases where the system can immediately present 1 item to the user using just 2 sentences, and the user then selects that item, i.e. Reward = $-(1.20 \times 1) + (.121 \times 100) - (2 \times 1.43) = 8.06$

9.5 Reinforcement Learning Experiments

We now formulate the problem as a Markov Decision Process (MDP), where states are NLG dialogue contexts and actions are NLG decisions. Each state-action pair is associated with a transition probability, which is the probability of moving from state s at time t to state s' at time $t+1$ after having performed action a when in state s. This transition probability is computed by the environment model (i.e. the user simulation and realiser), and explicitly captures the uncertainty in the environment. This is a major difference to other non-statistical planning approaches. Each transition is also associated with a reinforcement signal (or Reward) r_{t+1} describing how good the result of action a was when performed in state s. The aim of the MDP is to maximise long-term expected reward of its decisions, resulting in a *policy* which maps each possible state to an appropriate action in that state (see Chapter 3).

We treat IP as a hierarchical joint optimisation problem, where first the IP structure is chosen and then the number of attributes is decided, as shown in Figure 9.4. At each generation step, the MDP can choose 1-5 attributes (e.g. cuisine, price range, location, food quality, and/or service quality). Generation stops as soon as the user is predicted to select an item, i.e. the IP task is successful. (Note that the same constraint is operational for the WOZ baseline.)

$$\left[\begin{array}{ll}
\text{ACTION:} & \left[\textit{IP:} \left\{ \begin{array}{l} \text{SUMMARY} \\ \text{COMPARE} \\ \text{RECOMMEND} \end{array} \right\} \left\{ \text{attr: } 1\text{–}5 \right\} \right] \\
\text{STATE:} & \left[\begin{array}{l}
\texttt{attributes:} \left\{ \texttt{1-15} \right\} \\
\texttt{sentence:} \left\{ \texttt{2-18} \right\} \\
\texttt{dbHitsFocus:} \left\{ \texttt{1-100} \right\} \\
\texttt{userSelect:} \left\{ \texttt{0,1} \right\} \\
\texttt{userAddInfo:} \left\{ \texttt{0,1} \right\} \\
\texttt{userElse:} \left\{ \texttt{0,1} \right\}
\end{array} \right]
\end{array}\right]$$

Fig. 9.4 State-Action space for the RL-NLG problem

States are represented as sets of NLG dialogue context features. The state space comprises "low-level" features about the realiser behaviour (two discrete features representing the number of attributes and sentences generated so far) and three binary features representing the user's predicted next action, as well as "high-level" features provided by the DM (e.g. current database hits in focus). We trained the policy using the SHARSHA algorithm (Shapiro and Langley, 2002) with linear function approximation (Sutton and Barto, 1998), and the simulation environment described in section 9.4. The policy was trained for 60,000 iterations.

9.5.1 Experimental Set-up

We compare the learned strategies against the *WOZ baseline* as described in section 9.3.3. For attribute selection we choose a majority baseline (randomly choosing between 3 or 4 attributes) since the attribute selection models learned by Supervised Learning on the WOZ data didn't show significant improvements.

For training, we used the user simulation model most similar to the data, see section 9.4.1. For testing, we test with the *different* user simulation model (the one which is most dissimilar to the data).

We first investigate how well IP structure (without attribute choice) can be learned in increasingly complex generation **scenarios**. A generation scenario is a combination of a particular kind of surface realiser (template vs. stochastic) along with access to certain features of the dialogue context. We therefore investigate the following cases:

1.1. IP structure choice, Template realiser: Predicted next user action varies according to bi-gram model; Number of sentences and attributes per IP strategy is set by defaults, reflecting a template-based realiser.

1.2. IP structure choice, Stochastic realiser: IP structure where number of attributes per NLG turn is given at the beginning of each episode (e.g. set by the DM); Sentence generation according to the SPaRKy stochastic realiser model as described in section 9.3.2.

We then investigate different scenarios for *jointly* optimising IP structure (IPS) and attribute selection (Attr) decisions.

2.1. IPS+Attr choice, Template realiser: Predicted next user action varies according to tri-gram (i.e. IPS + Att) model; Number of sentences per IP structure set to default.

2.2. IPS+Attr choice, Template realiser+Focus model: Tri-gram user simulation with Template realiser and Focus of attention model with respect to #DBhits and #attributes as described in section 9.4.2.

2.3. IPS+Attr choice, Stochastic realiser: Tri-gram user simulation with sentence/attribute relationship according to Stochastic realiser as described in section 9.3.2.

2.4. IPS+Attr choice, Stochastic realiser+Focus model: i.e. the full model = Predicted next user action varies according to tri-gram model+ Focus of attention model + Sentence/attribute relationship according to stochastic realiser.

9.5.2 Results

We compare the average final reward (see equation 9.1) gained by the baseline against the trained RL policies in the different scenarios for each 1000 test runs, using a paired samples t-test. The results are shown in Table 9.3. In 5 out of 6 scenarios

Scenario	Wizard Baseline average Reward	RL average Reward	RL % - Baseline % = % improvement
1.1	-15.82(±15.53)	-9.90***(±15.38)	**89.2%** - 85.6%= 3.6%
1.2	-19.83(±17.59)	-12.83***(±16.88)	**87.4%** - 83.2%= 4.2%
2.1	-12.53(±16.31)	-6.03***(±11.89)	**91.5%** - 87.6%= 3.9%
2.2	-14.15(±16.60)	-14.18(±18.04)	**86.6%** - 86.6%= 0.0%
2.3	-17.43(±15.87)	-9.66***(±14.44)	**89.3%** - 84.6%= 4.7%
2.4	-19.59(±17.75)	-12.78***(±15.83)	**87.4%** - 83.3%= 4.1%

Table 9.3 Test results for 1000 dialogues, where *** denotes that the RL policy is significantly ($p < .001$) better than the Baseline policy

the RL policy significantly ($p < .001$) outperforms the human wizard baseline. We also report on the percentage of the top possible reward gained by the individual policies, and the raw percentage improvement of the RL policy. Note that the best possible (100%) reward can only be gained in rare cases (see section 9.4.3). An overview of the different IP strategies learned for each setup can be found in Table 9.4.

The learned RL policies show that lower level features are important in gaining significant improvements over the baseline. The more complex the scenario, the harder it is to gain higher rewards for the policies in general, but the relative improvement in rewards also increases with complexity: the baseline does not adapt well to the variations in lower level features whereas RL learns to adapt to the more challenging scenarios.

For example, the RL policy for scenario 1.1 learned to start with a SUMMARY if the initial number of items returned from the database is high (>30). It will then stop generating if the user is predicted to select an item. Otherwise, it continues with a RECOMMEND. If the number of database items is low, it will start with a COMPARE and then continue with a RECOMMEND, unless the user selects an item. Also see Table 9.4. Note that the WOZ strategy behaves as described in Figure 9.3.

In addition, the RL policy for scenario 1.2 learns to adapt to a more complex scenario: the number of attributes requested by the DM and produced by the stochastic sentence realiser. It learns to generate the whole sequence (SUMMARY+COMPARE +RECOMMEND) if *#attributes* is low (<3), because the overall generated utterance (final *#sentences*) is still relatively short. Otherwise the policy is similar to the one for scenario 1.1.

The RL policies for jointly optimising IP strategy and attribute selection learn to select the number of attributes according to the generation scenarios 2.1-2.4. For example, the RL policy learned for scenario 2.1 generates a RECOMMEND with 5 attributes if the database hits are low (<13). Otherwise, it will start with a SUMMARY using 2 attributes. If the user is predicted to narrow down his focus after the SUMMARY, the policy continues with a COMPARE using 1 attribute only, otherwise it helps the user by presenting 4 attributes. It then continues with RECOMMEND(5), and stops as soon as the user is predicted to select one item.

Scenario	strategies learned
1.1	RECOMMEND COMPARE COMPARE+RECOMMEND SUMMARY SUMMARY+COMPARE SUMMARY+RECOMMEND SUMMARY+COMPARE+RECOMMEND.
1.2	RECOMMEND COMPARE COMPARE+RECOMMEND SUMMARY SUMMARY+COMPARE SUMMARY+RECOMMEND SUMMARY+COMPARE+RECOMMEND.
2.1	RECOMMEND(5) SUMMARY(2) SUMMARY(2)+COMPARE(4) SUMMARY(2)+COMPARE(1) SUMMARY(2)+COMPARE(4)+RECOMMEND(5) SUMMARY(2)+COMPARE(1)+RECOMMEND(5)
2.2	RECOMMEND(5) SUMMARY(4) SUMMARY(4)+RECOMMEND(5)
2.3	RECOMMEND(2) SUMMARY(1) SUMMARY(1)+COMPARE(4) SUMMARY(1)+COMPARE(1) SUMMARY(1)+COMPARE(4)+RECOMMEND(2)
2.4	RECOMMEND(2) SUMMARY(2) SUMMARY(2)+COMPARE(4) SUMMARY(2)+RECOMMEND(2) SUMMARY(2)+COMPARE(4)+RECOMMEND(2) SUMMARY(2)+COMPARE(1)+RECOMMEND(2)

Table 9.4 RL strategies learned for the different scenarios, where (n) denotes the number of attributes generated

The learned policy for scenario 2.1 generates 5.85 attributes per NLG turn on average (i.e. the cumulative number of attributes generated in the whole NLG sequence, where the same attribute may be repeated within the sequence). This strategy primarily adapts to the variations from the user simulation (tri-gram model). For scenario 2.2 the average number of attributes is higher (7.15) since the number of attributes helps to narrow down the user's focus via the DBhits/attribute relationship specified in section 9.4.2. For scenario 2.3 fewer attributes are generated on average (3.14), since here the number of attributes influences the sentence realiser, i.e. fewer attributes results in fewer sentences, but does not impact the user's focus. In scenario 2.4 all the conditions mentioned above influence the learned policy. The average number of attributes selected is still low (3.19).

In comparison, the average (cumulative) number of attributes for the WOZ baseline is 7.10. The WOZ baseline generates all the possible IP structures (with 3 or 4 attributes) but is restricted to use only "high-level" features (see Figure 9.3). By beating this baseline we show the importance of the "low-level" features. Nevertheless, this wizard policy achieves up to 87.6% of the possible reward on this task, and so can be considered a serious baseline against which to measure performance.

The only case (scenario 2.2) where RL does not improve significantly over the baseline is where lower level features do not play an important role for learning good strategies: scenario 2.2 is only sensitive to higher level features (DBhits).

9.6 Evaluation with real users

The policy described above was then deployed in an extensive online user study, involving 131 users and more than 800 test dialogues, which explores its contribution to overall 'global' task success in a spoken dialogue system.

The IP policy was integrated into the 'CamInfo' system, a spoken dialogue system developed in the CLASSiC project[4] providing tourist information for Cambridge (Young et al, 2009). This baseline system was made accessible by phone using VoIP technology, enabling out-of-lab evaluation with large numbers of users. The speech recogniser (ASR), semantic parser (SLU) and dialogue manager (DM) were all developed at Cambridge University. For speech synthesis (TTS), the Baratinoo synthesiser, developed at France Telecom, was used.

The DM uses a POMDP (Partially Observable Markov Decision Process) framework, allowing it to process N-Best lists of ASR hypotheses and keep track of multiple dialogue state hypotheses. The DM policy is trained to select system dialogue acts given a probability distribution over possible dialogue states. It has been shown that such dialogue managers can exploit the information in the N-Best lists (as opposed to only using the top ASR hypothesis) and are therefore particularly effective in noisy conditions (Young et al, 2009).

The Natural Language Generation component of this baseline system is a standard rule-based surface realiser covering the full range of system dialogue acts that the dialogue manager can produce. It has only one IP strategy, i.e., the system only provides information about database entries in the form of single venue recommendations (the RECOMMEND strategy, see Table 9.1). The attributes of the venue to be presented are selected heuristically. In the extended version of the system, the IP strategy is replaced by our trained NLG component, which is optimised to decide between different IP actions as described above.

[4] www.classic-project.org

The subjects were directed to a webpage with detailed instructions and for each task, a phone number to call, and the scenario to follow. A scenario describes a place to eat in town, with some constraints, for example: *"You want to find a moderately priced restaurant and it should be in the Riverside area. You want to know the address, phone number, and type of food."* After the dialogue, the subjects were asked to fill in a short questionnaire.

For the objective evaluation of the two systems we focused on measuring task completion rates. A full presentation and discussion of the results is given in (Rieser et al, 2011). In summary, we found that the trained Information Presentation strategy significantly improves dialogue task completion for real users, with up to a 9.7% increase (30% relative) compared to the deployed dialogue system which uses conventional, hand-coded presentation prompts. This result establishes the benefits of this overall methodology for building NLG systems.

9.7 Conclusion

This Chapter has illustrated that the framework developed throughout the book can be applied to a new set of problems in NLG, thereby providing evidence for the generality of our techniques.

We presented a new data-driven method for Information Presentation in Spoken Dialogue Systems using a statistical optimisation framework for content structure planning and attribute selection. This work is the first to apply a data-driven optimisation method to the IP decision space, and to show the utility of both lower-level and higher-level features for this problem.

We collected data in a Wizard-of-Oz (WOZ) experiment and showed that human "wizards" mostly pay attention to 'high-level' features from Dialogue Management. The WOZ data was used to build statistical models of user reactions to IP strategies, and a data-driven reward function for RL. We compared the observed human behaviour (the 'human wizard baseline') against policies optimised using Reinforcement Learning (RL), in a variety of scenarios. Our optimised policies significantly outperform the IP structuring and attribute selection present in the WOZ data, especially when performing in complex generation scenarios which require adaptation to, e.g. number of database results, utterance length, etc. While the human wizards were able to attain up to 87.6% of the possible reward on this task, the RL policies are significantly better in 5 out of 6 scenarios, gaining up to 91.5% of the total possible reward.

We have also shown that adding predictive "lower level" features, e.g. from the surface realiser and a user reaction model, is important for learning optimal IP strategies. Future work could also include the predicted TTS quality (Boidin et al, 2009) as a feature.

The method presented here was also evaluated with real users (Rieser et al, 2011), where we found that the trained Information Presentation strategy significantly improves dialogue task completion for real users, with up to a 9.7% increase over a state-of-the-art baseline IP strategy.

This methodology provides new insights into the nature of the IP problem, which has previously been treated as a module following dialogue management with no access to lower-level context features. The data-driven planning method applied here promises a significant upgrade in the performance of generation modules, and thereby of Spoken Dialogue Systems in general.

Chapter 10
Conclusion

Fig. 10.1 Two chatbots interacting (Cornell Creative Machines Lab)

In this final Chapter we discuss what further challenges machines need to meet when learning how to engage in useful, meaningful, and natural dialogues with humans. In contrast to the approach adopted by many "chatbots" (see Figure 10), which typically use shallow pattern-matching techniques to retrieve a next system move, we argue that coherent dialogue actions can only be chosen when the system knows the likely meanings of its dialogue contributions, and has a representation of dialogue states which allows it to compute the utility of each of those moves. This is the approach that we have focussed on in this book, for building useful task-based dialogue systems, and it is a topic of research worldwide. There is still some distance

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6_10,

to travel, however, in developing more sophisticated systems using these methods, and in developing more generic tools for creating statistical dialogue systems.

In this Chapter we will first summarise the main contributions of this book to this overall research programme, and then identify some key directions for future research.

10.1 Contributions

In the area of dialogue strategy development, machine learning methods, such as Reinforcement Learning (RL), have recently been explored as an alternative method to hand-coding dialogue strategies. One of the key advantages of such statistical optimisation methods for dialogue strategy design is that the problem can be formulated as a principled mathematical model model which can be optimised by training on real data. In Chapter 4 we have demonstrated that RL-based strategies can outperform hand-coded strategies with manually tuned thresholds for a wide range of application scenarios (also see (Rieser and Lemon, 2007, 2008b)). In cases where a system is designed from scratch, however, there is often no suitable in-domain data. Collecting dialogue data without a working prototype is problematic, leaving the developer with a classic chicken-and-egg problem.

In Chapter 5 we proposed a method for learning dialogue strategies by using simulation-based RL, where the simulated environment is learned from small amounts of Wizard-of-Oz (WOZ) data. The WOZ data collection was described in Chapter 6 (also see (Rieser et al, 2005)). Using WOZ data rather than data from real Human-Computer Interaction allows us to learn optimal strategies for domains where no working dialogue system already exists. To date, automatic strategy learning has been applied to dialogue systems which have already been deployed in the real world using handcrafted strategies. In such work, strategy learning was performed based on already present extensive online-operation experience, e.g. (Henderson et al, 2005, 2008; Singh et al, 2002). In contrast to this preceding work, our approach enables strategy learning in domains where no prior system is available. Optimised learned strategies are then available from the first moment of online operation, and handcrafting of dialogue strategies is avoided. This independence from large amounts of in-domain dialogue data allows researchers to apply RL to new application areas beyond the scope of existing dialogue systems. We called this method ‘bootstrapping’.

In Chapter 7 we applied this framework to optimise multimodal information-seeking dialogue strategies in a music player domain. Dialogue Management and Natural Language Generation are two closely interrelated problems for information seeking dialogues: the decision of *when* to present information depends on *how many* pieces of information to present and the available options for *how* to present them, and vice versa. We therefore formulate the problem as a hierarchy of joint learning decisions which are optimised together (Rieser and Lemon, 2008d, 2011; Rieser et al, 2011).

The learned strategy was first evaluated in simulation (Chapter 7), and we also tested its performance with real users, as described in Chapter 8. In particular, we compared the performance of the RL-based strategy against a supervised learning (SL) strategy which is obtained from (human) wizards' policies from the original data (Rieser and Lemon, 2006c). This comparison allowed us to measure relative improvement over the initial strategies contained in the training data. Our results showed that RL significantly outperforms SL when interacting in simulation as well as for interactions with real users (Rieser and Lemon, 2008c, 2011). The RL-based policy gained on average 50-times more reward when tested in simulation, and almost 18-times more reward when interacting with real users. Users also subjectively rated the RL-based policy on average 10% higher. We post-evaluated the simulated learning environment by comparing different aspects of the 3 corpora gathered so far: the WOZ study, the dialogues generated in simulation, and the final user tests. We showed that results obtained in simulation are comparable to results for real users. Hence, we concluded that a strategy bootstrapped from WOZ data is transferable to real Human-Computer Interaction (Rieser and Lemon, 2008c).

In Chapter 9 we showed that our method can be used for new problems – in this case the Natural Language Generation task of Information Presentation (Lemon et al, 2012; Liu et al, 2009; Rieser and Lemon, 2009a, 2010a; Rieser et al, 2010, 2011).

One major advantage of RL-based dialogue strategy development is that the same objective function can be used for dialogue evaluation, as well as for optimisation. Despite its central role for RL, it has received little attention so far. In most of the previous research it is manually set (Paek, 2006). One focus of this book was to optimise dialogue strategies with respect to real user preferences. We constructed our objective function from the WOZ data using a modified version of the PARADISE framework which estimates subjective user preferences from objective dialogue measures (Walker et al, 1997). In the post-evaluation we tested different aspects of this objective function, such as its replicability on different data sets and its prediction accuracy (Rieser and Lemon, 2008a). We also experimented with non-linear rewards for strategy learning.

10.2 Discussion

We have already discussed some detailed directions for future work in individual chapters. At this point, we discuss two of the more general claims brought forward by this book. First, we address some of the limitations of the bootstrapping approach and review our "lessons learned". Second, we discuss to what extent RL-based strategy development is suitable for industrial applications.

10.2.1 Lessons Learned

In this book we demonstrate that one can "bootstrap" an optimised strategy from limited amounts of WOZ data using simulation-based RL. Our results show that such a RL-based strategy significantly improves over the strategies derived from the initial corpus, and that results obtained in simulation do carry over to tests with real users. We therefore conclude that a simulated learning environment which is bootstrapped from WOZ data can be a valid and useful estimate of real HCI.

So far, we have only shown this for one specific example. It is not clear how generally applicable this approach is to other strategy learning problems for dialogue systems and to other WOZ data sets. From our limited evidence we are not yet able to make any concrete statements about the required *quality* and *quantity* of the WOZ data with respect to the complexity of the learning problem. Nevertheless, from our "lessons learned" we can give some recommendations on these issues.

10.2.1.1 Required Data Quality

In Chapter 8.5 we compared several aspects of the WOZ corpus, the simulated corpus, and the corpus gathered with real users. We demonstrated that some of the experimental conditions between WOZ and real setup only need to be similar, rather than identical, since strategies learned for a particular set of conditions can transfer to a different set of conditions under certain circumstances. In particular, we showed that a strategy trained on a higher noise level transfers well to situations with lower noise, confirming results by Lemon and Liu (2007).

However, other conditions might not transfer well and thus have to be correctly reflected in the initial data set. The results by Lemon and Liu (2007), for example, indicate that a policy trained on different user types (cooperative vs. uncooperative user) do not produce compatible results. This suggests that a "good enough" user model has to be learned from data for the policy to fit the intended application domain.

From our 'lessons learned' we conclude that for learning output presentation strategies it is important to keep the output media constant across experiments. In particular, the size of the screen influences how the number of multimodal items is evaluated by the user, and the TTS quality influences how verbal presentation strategies are evaluated (cf. user comments in Section 8.3.1.3).

10.2.1.2 Required Data Quantity

In Section 7.1.1 we outlined the problem of how much data is enough for bootstrapping a simulated learning environment. Most of the simulated components are constructed using a form of Supervised Learning (or a frequency-based approach such as the bi-gram user simulation). For SL problems there is a "triple trade-off problem" (Dietterrich, 2003) between the accuracy required, complexity of the learning problem, and the available data. In Section 7.1.1 we discussed that under certain circumstances one can cut back on accuracy for the benefit of exploring larger policy spaces (also see results by Ai et al (2007b)).

The purpose of simulated learning environments is two-fold: they should ideally cover all possible situations or states that the dialogue can be in, and they should give distinctive feedback to the learner about the 'desirability' of being in a particular state (with respect to the expected final reward), see Section 5.2.1. In particular we found that all components need to guarantee full coverage of possible situations, while the feedback can be distributed over several components.

For example, in Section 7.8 we discuss different techniques for creating user simulations with full coverage of the defined context. One of these user simulations is "cluster-based", i.e. it clusters together different states (see Section 7.8.4), causing the simulated user to react similarly in similar situations. However, the reward function and the implemented domain constraints provide distinctive feedback to the learner so that it can experience the difference between taking different actions (see the discussion in Section 7.8.4.2).

Recent research also investigates rapid learning from small data sets, e.g. (Crook and Lemon, 2011b; Gasic and Young, 2011; Pietquin et al, 2011b), where no simulated environment is required. In this work the the amount of data needed for each algorithm is also evaluated.

10.2.2 RL for Commercial Dialogue Strategy Development

In this book we have discussed different machine learning techniques as a possibility for bridging the gap between industry and research (see Section 2.2.4). However, statistical learning techniques, especially Reinforcement Learning, have been criticised for not being suitable for commercial development (Paek, 2006). The following points are mentioned in particular:

1. RL requires large amounts of annotated data, which limits the complexity of the learned strategy.
2. Many of the aspects are still hand-crafted, especially the selection of the state space and the objective function used for strategy optimisation.
3. The learned policy is a "black box" which is difficult for the system designer to change.
4. It is not clear whether results obtained in simulation transfer to interaction with real users. In particular, it is not clear whether the objective function (reward) used for strategy optimisation reflects the preferences of real users.

We have provided evidence that these points can be addressed without major difficulties:

1. We use simulation-based RL which automatically generates the data required for strategy learning. The simulated learning environment can be obtained from a relatively small WOZ corpus, without the need for a working prototype.

2. We also show that it is possible to construct a learning environment which is completely data-driven, including state space selection and the design of the objective function.
3. The learned policy is transferred into a table look-up (Section 7.11.4), which allows the dialogue designer to inspect and (if required) change the policy.
4. Finally, we show that results obtained in simulation do transfer to interaction with real users. We also explicitly evaluate whether the objective function indeed reflects real user preferences.

In sum, this book provides counter-examples to these arguments against the commercial use of RL. Furthernore, in very recent work, RL approaches are now being deployed in commercial settings and tools, for example by researchers at AT&T (Williams, 2008, 2011) and France Telecom/Orange Labs (Putois et al, 2010).

10.3 Outlook: challenges for future statistical dialogue systems

Reinforcement Learning-based control strategies have been shown to be successful in a wide variety of fields, such as air traffic management, communication networks, water allocation, diagnosis based on medical images, and so on, see (Weber et al, 2008) for an overview. One of the best known examples is the application of RL to board games such as backgammon (Tesauro, 1995) or chess (Baxter et al, 2001). Chess was considered to be one of the last bastions of human intelligence, but in 1997 world champion Garry Kasparov lost a six-game match against IBM's chess computer "Deep Blue".[1] According to its developers several factors contributed to this success, such as strong computing power, an emphasis on search extensions (i.e. exploration), and a complex evaluation function (Campbell et al, 2002). In particular Deep Blue's ability to break up blocked positions by applying very forward-looking strategies surprised chess experts.

However, one of Deep Blue's bottlenecks was that it was not a "learning system", i.e. it was not able to learn and adapt from his own successes and mistakes. Deep Blue was a product of thorough engineering, rather than being truly "intelligent". This mock intelligent behaviour evokes certain associations with its famous predecessor *The Turk*, designed by Wolfgang von Kempelen (see Chapter 6).

Dialogue is similar to chess in the sense that it is set in a temporal and dynamic environment (see comparison in Section 3.1). Nevertheless, participating in natural language dialogue is different from playing chess in several important ways, and we see some interesting challenges for future dialogue research driven by these differences.

First of all, in a game of chess all possible moves and board positions are definite – there is no uncertainty about which move has actually been made by the other participant. In dialogue the interpretation of the input is uncertain, which is especially

---

[1] see `www.research.ibm.com/deepblue/` (24. April 2008)

true for human-machine dialogue where ASR and SLU can introduce additional uncertainty. Therefore, there is a major direction for future research in learning how to handle this input uncertainty, as currently explored by several research groups using Partially Observable Markov Decision Processes (POMDPs), e.g. (Bohus, 2007; Crook and Lemon, 2010; Gasic and Young, 2011; Thomson et al, 2008; Williams, 2006, 2011).

Furthermore, a chess move has only one possible realisation. For the other participant it (usually) doesn't matter how moving a piece from position A to position B is actually performed. In dialogue interaction however, a speech action can be realised in various ways, and the other dialogue participant will adapt his/her behaviour dependent on the chosen utterance (Stenchikova and Stent, 2007). For this reason, statistical approaches to Natural Language Generation (NLG) for dialogue systems are an important ongoing research area e.g. (Dethlefs and Cuayahuitl, 2010; Dethlefs and Cuayáhuitl, 2011; Janarthanam and Lemon, 2008, 2010a,b; Lemon, 2008, 2011; Nakatsu and White, 2006; Oh and Rudnicky, 2002; Rieser and Lemon, 2010b; Rieser et al, 2010, 2011; Walker et al, 2001b). Here the overall aim is the same as in this book: to improve the global user experience by optimising sequences of local decisions.

We think that the bootstrapping approach from WOZ data may be especially promising here, as we consider humans to be experts in generating natural utterances. In contrast to high-level action selection in simulating behaviour for a dialogue system, talking to other people is a more natural task for humans (see discussion in Section 2.3.2).

Similarly, in chess, a move cannot usefully be broken down into smaller temporal components, but in spoken dialogue there are useful semantic and pragmatic components beneath the turn level. Another important new research area for spoken interaction is therefore the development of *incremental* systems (Schlangen and Skantze, 2011; Skantze and Schlangen, 2009). "Incremental" refers to the design goal that incoming speech should be decoded on a word-by-word basis (rather than at the utterance level, which is the current state-of-the-art), and also that systems should be able to plan and execute utterances at a sub-utterance level (Skantze and Hjalmarsson, 2010), for example by completing a user's utterance as in the example below:

```
User: I'm looking for Paranoid Android by umm <pause>
System: by Radiohead ?
User: Yes!
```

Such systems would be more natural and efficient than current SDS, but the requirement for incrementality generates serious new challenges for statistical approaches. In particular, incremental input processing clearly generates more system states with higher frequency, thereby increasing the size of the state spaces required for statistical dialogue models. Likewise, incremental system output introduces many new decisions and trade-offs concerned with generating utterance fragments and feedback signals at the right time. The resulting learning problems will require new techniques such as state-space compression and rapid, sample-efficient learning algorithms (Crook and Lemon, 2011b,b; Gasic et al, 2010; Pietquin et al, 2011b). Efficient learning methods also hold the promise of learning online from small data-sets, and so constitute a potential alternative to the method that we have developed in this book.

A further vital direction for the field is in testing learned policies “in the wild”, see for example (Gruenstein and Seneff, 2007): with real users who have their own genuine goals – rather than in laboratory-based experiments as is the current state-of-the-art. This is similar to testing Deep Blue against Gary Kasparov, who certainly pursued a genuine personal goal in attempting to win the match. Recent advances in Voice-Over IP capabilities of mobile devices now make “in the wild” testing quite practical, by opening up systems for free public use, and first steps in this direction have been made in the evaluation of the CLASSiC project[2] systems, for Tourist Information and Appointment Scheduling, and also within the Spoken Dialogue Challenge (Black et al, 2011).

Finally, looking further ahead to the Natural Language interfaces of the future, we note a requirement for processing of non-cooperative, or “strategic”, dialogue (Asher et al, 2011). While chess is a non-cooperative interaction (in that players have incompatible goals), all current dialogue interfaces are strictly cooperative. That is, the system’s goal (as represented in our work by the reward function) is to help the user to complete their tasks in an efficient manner. However, some dialogue tasks (for example negotiation) will require systems to pursue their own domain goals to some extent (i.e. using a reward function not wholly determined by user preferences), and to recognise that some user utterances are not fully cooperative (for example, information may be witheld while bargaining). Again, such requirements create interesting challenges for our models and will drive further progress in statistical approaches to meaningful human communication.

[2] See www.classic-project.org

Appendix A
Example Dialogues

A.1 Wizard-of-Oz Example Dialogues

V. Rieser and O. Lemon, *Reinforcement Learning for Adaptive Dialogue Systems*,
Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-642-24942-6,

Table A.1 Example dialogue from the SAMMIE WOZ corpus (mixed wizard strategy)

s1	User:	Bitte suche das Lied ‚Layla'.
	Transl.:	*Search for the song 'Layla' please.*
	SA* user:	`add --> song-title(Layla)`
	Wizard:	Wir haben das Lied gefunden. Es existiert 142 mal in verschiedenen Ausführungen
	Transl.:	***The song is retrieved. There are 142 different versions.***
	mmAction:	[Table with 142 entries is shown on the screen.]
	SA wizard:	`presentMM --> song-title (db:142)`
s2	User:	Bitte suche das Lied on dem Komponisten ‚Eric Clapton'.
	Transl.:	*Search for the song by the composer 'Eric Clapton' please.*
	mm Action:	[User does not react to what is displayed.]
	SA user:	`change --> artist('Eric Clapton')`
	Wizard:	Es gibt das Lied von ‚Eric Clapton' oder von 'Derek & the Dominos'.
	Transl.:	***The song exists by 'Eric Clapton' or by 'Derek & the Dominos'.***
	SA wizard:	`presentVerbal -> song-title, artist` **(db:2)**
s3	User:	Bitte spiele das Lied von ‚Eric Clapton Unplugged' ab.
	Transl.:	*Play the song from 'Eric Clapton Unplugged' please.*
	SA user:	`repeat -->album('Eric Clapton Unplugged')`
	Wizard:	Ich spiele das Lied jetzt ab.
	Transl.:	*OK I am going to play the song now.*
	SA wizard:	`feedback/others`
	mmAction:	[music]
	user ratings	taskEase: 2, timing:2, mmPresent.: 1 , verbalPresent: 3, futureUse:3

* SA = Speech Act

Comments:

In this example the wizard uses a mixed strategy presenting information multimodally in turn 1 and verbally in turn 2. The first time 142 items are presented multimodally by verbally reporting the number of retrieved items while displaying them on the screen. The database used in the WOZ experiments contained over a million songs and is searched with partial string match. Therefore the retrieved items for 'Layla' can be as high as 142. The user does not directly react to what is shown on the screen (either by referring to it or clicking), but continues providing information until the wizards (verbally) presents two alternatives, where the user indirectly chooses one by providing an album name which also disambiguates the artist. The user rankings from the questionnaire are about average

Table A.2 Example dialogue from the SAMMIE WOZ corpus (wizard not starting the presentation phase early enough).

s1	User:	Also, ich suche das Lied ‚Smile'.
	Transl.:	*Well, I am searching for the song 'Smile'.*
	SA user:	`add --> song-title(Smile)`
	Wizard:	Welches Lied suchst du?
	Transl.:	*What song are you searching for?*
	comment:	[Song name was deleted]
	SA wizard:	`reject(song-title)`
s2	User:	Smile.
	Transl.:	*Smile.*
	SA user:	`repeat --> song-title(Smile)`
	Wizard:	Du suchst das Lied ‚Smile'?
	Transl.:	*Are you searching for the song 'Smile'?*
	SA wizard:	`explConf(song-title)`
s3	User:	Ja, Smile ja.
	Transl.:	*Yes, Smile yes.*
	SA user:	`y-answer --> song-title(Smile)`
	Wizard:	Ich suche es... Ich habe vier Lieder mit dem einfachen Titel ‚Smile'.
	Transl.:	*I am searching... I found four songs with only 'Smile' as the title.*
	SA wizard:	`presentVerbal(song-title)`
s4	User:	Ja denn, das Lied bitte anzeigen.
	Transl.:	***Very well then, why don't you show it?***
	SA user:	`others`
	Wizard:	OK.
	Transl.:	*OK.*
	SA wizard:	`presentMM(song-title)`
	user ratings	taskEase: 2, timing: 2, mmPresent.: 5, verbalPresent: 3, futureUse:3

Comments:

In this example the user is asking to see a short list on the screen (after resolving some understanding errors induced by the deletion tool), which is indicating that (a) the wizard did not enter the presentation phase early enough, and (b) that for the user 4 items seems to be a reasonable number to display. The user scores reflect this fact: the presentation mode got a maximal score (which will influence learning towards showing short lists), whereas the task ease got a lower score (task ease is used as an overall reward, i.e. this 'tells' the learner to enter the presentation mode earlier).

A.2 Example Dialogues from Simulated Interaction

Table A.3 Example dialogue with the SL policy operating in the simulated environment.

s1	SL policy:	`greet (db:438)`
	sim.User:	`add-->genre(Rock)`
	state:	`filled [slot1], confirmed [], db:142`
s2	SL policy:	`implConfAskASlot --> genre, artist)`
	sim.User:	`repeat-->genre(Rock)`
	state:	`filled [slot1], confirmed[slot1], db:142`
s3	SL policy:	`present[mm] -->genre`
	sim.User:	`change-->artist(Jimi Hendrix)`
	state:	`filled [slot1,slot2], confirmed [slot1], db:4`
s4	SL policy:	`implConfAskASlot-->artist, album`
	sim.User:	`add-->album(Experience Hendrix: The Best of Jimi Hendrix)`
	state:	`filled [slot1,slot2,slot3], confirmed [slot1,slot2], db:4`
s5	SL policy:	`implConfAskASlot-->album, song-title`
	sim.User:	`add-->song-title(Voodoo Child)`
	state:	`filled [slot1,slot2,slot3,slot4], confirmed [slot1,slot2,slot3], db:1`
s6	SL policy:	`explicitConfirm-->song-title`
	sim.User:	`repeat-->song-title(Voodoo Child)`
	state:	`filled [slot1,slot2,slot3,slot4], confirmed [slot1,slot2,slot3,slot4], db:1`
s7	SL policy:	`present[verbal] -->genre, artist, album, song-title`
	sim.User:	`yes-answer-->yes(yes)`
	state	`filled [slot1,slot2,slot3,slot4], confirmed [slot1,slot2,slot3,slot4], db:1`
	reward:	$lengthPenalty: -141.4, complVal: 118, presentVal: 8.7 \times (4.33 - 2151.23)$ finalReward:-18701.43

Comments:

This example shows a dialogue where the SL policy is operating in the simulated environment. Presentation timing and amount of presented information are not ideal (and therefore leading to quite a long dialogue): in turn 2 the system presents a list of 238 items, whereas when only having 4 alternatives (from turn 3 onwards) it fails to show a list.

Table A.4 Example dialogue with the RL policy operating in the simulated environment (mixed strategy).

s1	RL policy:	`greet (db: 438)`
	sim.User:	`add-->artist(Nirvana)`
	state:	`filled [slot2], confirmed [], db:26`
s2	RL policy:	`implConfAskASlot --> artist, album)`
	sim.User:	`add-->album(MTV Unplugged)`
	state:	`filled [slot2,slot3], confirmed [slot2], db:14`
s3	RL policy:	`present[mm] -->artist, album`
	sim.User:	`click-->song-title(On a Plain),genre(Alternative)*`
	state:	`filled [slot1,slot2,slot3,slot4], confirmed [slot2,slot3], db:1`
s4	RL policy:	`present[verbal] -->artist, album, song-title`
	sim.User:	`yes-answer-->yes(yes)`
	state:	`filled [slot1,slot2,slot3,slot4], confirmed [slot1,slot2,slot3,slot4], db:1`
	reward:	$lengthPenalty: -80.8, complVal: 118, presentVal: 8.7 \times (4.33 + 7.23)$ finalReward:140.3

Comments:
This example shows the RL policy interacting in the simulated environment. The system follows a mixed strategy: first presenting a list of 14 items on the screen and then presenting a unique item verbally. * the user acts `add` after a multimodal presentation are equivalent to clicking

Table A.5 Example dialogue with the RL policy operating in the simulated environment: presenting alternatives.

s1	RL policy:	`greet (db: 438)`
	sim.User:	`add-->artist(Bob Dylan)`
	state:	`filled [slot2], confirmed [], db:56`
s2	RL policy:	`implConf(artist)AskASlot(genre)`
	sim.User:	`change-->song-title(Hurricane)`
	state:	`filled [slot2,slot4], confirmed [slot2], db:2`
s3	RL policy:	`present[verbal] -->album1,album2`
	sim.User:	`add-->album(Desire)`
	state:	`filled [slot2, slot3, slot4], confirmed [slot2,slot4], db:1`
s4	RL policy:	`present[verbal] -->artist, album, song-title`
	sim.User:	`yes-answer-->yes(yes)`
	state:	`filled [slot2, slot3, slot4], confirmed [slot2, slot3, slot4], db:1`
	reward:	$lengthPenalty: -80.8, complVal: 118, presentVal: 8.7 \times (4.327 + 3.395)$ finalReward: 104.38

Comments:
This example shows the RL policy interacting in the simulated environment. The system presents two alternatives and a single item using only verbal output.

A.3 Example Dialogues from User Testing

Table A.6 Example dialogue with the SL policy operating with real users.

s1	SL system:	Hi, this is iTalk your talkative music player. How may I help you today?
	SA system:	`greet (db: 438)`
	User:	i would like to listen to a blues song
	SA user:	`add-->genre(Blues)`
	state:	`filled [slot1], confirmed [], db:175`
s2	SL system:	There are 175 items matching your query. A list of possible candidates are shown on the screen.
	SA system:	`presentMM--> genre (db:175.0)`
	User:	—
	SA user:	`nothing` (*time out*)
	state:	`filled [slot1], confirmed [], db:175`
s3	SL system:	A Blues song. By which artist?
	SA system:	`implConf(genre), askASlot(artist)`
	User:	eric clapton
	SA user:	`add-->artist('Eric Clapton) (db:25.0)`
	state:	`filled [slot1, slot2], confirmed [slot1], db:25`
s4	SL system:	Searching for music by Eric Clapton. Which album?
	SA system:	`implConf('Eric Clapton'), askASlot(album)`
	User:	mtv unplugged
	SA user:	`add-->album('MTV unplugged')`
	state:	`filled [slot1, slot2, slot3], confirmed [slot1, slot2], db:13`
s5	SL system:	A song from the album MTV unplugged. What is the title of the song?
	SA system:	`implConf(album), askASlot(song-title)`
	User:	what do you have
	SA user:	`request-help`
	state:	`filled [slot1, slot2, slot3], confirmed [slot1, slot2], db:13`
s6	SL system:	You have several options. For example: Layla , Nobody Knows You When You Are Down and Out , Lonely Stranger , Old Love , and San Francisco Bay Blues. Do you want to hear more?
	SA system:	`give-help`
	User:	no layla please
	SA user:	`add-->song-title(Layla)`
	state:	`filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot2, slot3], db:1`
s7	SL system:	You will now hear Layla by Eric Clapton from the album MTV unplugged. Are you happy with this option?
	SA system:	`presentVerbal-->genre,artist,album,song-title`
	User:	yes
	SA user:	`yes-answer`
	state:	`filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot2, slot3, slot4], db:1`
	reward:	$lengthPenalty: -141.4, complVal: 118, presentVal: 8.7 \times (4.33 - 7.63)$ finalReward: -6577.03
	user ratings	taskEase:2, timing:2, mmPresent:1, verbalPresent:2, futureUse:2

Comments:

This example shows the SL policy applied with real users. In turn 2 the user ignores the 175 items displayed, then the system does not display the 13 retrieved items. The user asks for help.

Table A.7 Example dialogue with the RL policy operating with real users: multimodal and verbal presentation

s1	RL system:	Hi. this is iTalk your talkative music player. How may I help you today?
	SA system	`greet (db:438)`
	User:	play me some blues
	SA user:	`add-->genre(Blues) (db:175)`
	state:	`filled [slot1], confirmed [], db:175`
s2	RL system:	Blues for type of music. By which artist?
	SA system:	`implConf(genre),askASlot(artist)`
	User:	from the album desire
	SA user:	`change-->album(Desire)(db:10)`
	state:	`filled [slot1, slot3], confirmed [slot1], db:10`
s3	RL system:	There are 10 items matching your query. A list of possible candidates are shown on the screen.
	SA system:	`presentMM-->genre,album`
	User:	[clicks]
	SA user:	`click->genre,song-title`
	state:	`filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot3], db:1`
s4	RL system:	You selected an item on the screen. Please wait a few moments. [...] You will now hear Mozambique by Bob Dylan from the album Desire. Are you happy with this option
	SA	`presentVerbal-->genre,album,artist,song-title`
	User:	yes
	SA	`yes-answer`
	state:	`filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot2, slot3, slot4], db:1`
	reward:	$lengthPenalty : -60.6, complVal : 118, presentVal : 8.7 \times (4.33 + 6.57)$ finalReward: 152.16
	user ratings	taskEase:6, timing:7, mmPresent.:7 verbalPresent: 6 futureUse:6

Comments:

This example shows the RL policy interacting with real users. The system uses a mixed strategy, showing multimodal output as soon as the number of retrieved items is below 15 (and therefore the reward will be positive), and presenting one item verbally.

Table A.8 Example dialogue with the RL policy operating with real users: verbal presentation of options

s1	RL system:	Hi. this is iTalk your talkative music player. How may I help you today?
	SA system:	`greet (db:438)`
	User:	i want to hear polly
	SA user:	`add-->song-title(Polly)`
	state:	`filled [slot4], confirmed [], db:2`
s2	RL system:	There are 2 results matching your query all by the artist Nirvana all from the genre Alternative all having the song title Polly, but a different album. You have the following options: Nevermind and MTV Unplugged. Which album would you like?
	SA system:	`presentVerbal-->song-title,artist,genre`
	comment:	*systems fills unique slots.*
	User:	mtv unplugged
	SA user:	`add-->album('MTV unplugged') (db:1)`
	state:	`filled [slot3, slot4], confirmed [song4], db:1`
s3	RL system:	There is only one result matching your query. Please wait a few moments. [...] You will now hear Polly by Nirvana from the album MTV unplugged. Are you happy with this option ?
	SA system:	`presentVerbal-->song-title,artist,genre,album`
	User:	yes
	SA user:	`yes-answer`
	state:	`filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot2, slot3, slot4], db:1`
	reward:	$lengthPenalty : -60.6, complVal : 118, presentVal : 8.7 \times (4.33 + 3.4)$ finalReward: 124.6
	user ratings	taskEase:7, timing:7, mmPresent.:7 verbalPresent: 7 futureUse:6

Comments:

This example shows the RL policy interacting with real users. Two alternatives are presented verbally.

Appendix B
Learned State-Action Mappings

The table in this section presents the learned state action mappings, and is to be read as follows. The first two columns constitute the state space. The first column shows the slots that have been filled and/or confirmed. The slots are:

slot 1: genre
slot 2: artist
slot 3: album
slot 4: song title

The second column is a represent possible numbers of database hits. Note that only a the possible number of items returned from the database is constrained by the structure of the task (i.e. how combinations of different slots values constrain the search space).

The third column is the optimal action for that state. The “x”s in the second column denote numbers of database hits that share the same optimal action (given the set of filled and confirmed slots). Horizontal lines are drawn between sets of states with different filled slots.

Learned State Action Mappings

state space																												system action
slot values	**no. DB hits**																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	18	20	25	26	34	40	54	56	92	142	175	438	
filled [], confirmed []																											x	askASlot
filled [slot1], confirmed []																				x					x	x		implConf
filled [slot1], confirmed [slot1]																				x					x	x		askASlot
filled [slot2], confirmed []																		x	x	x			x	x	x			implConf
						x							x	x														presentMM
filled [slot2], confirmed [slot2]																		x	x	x			x	x	x			askASlot
						x							x	x														presentMM
filled [slot3], confirmed []															x	x	x					x						implConf
				x	x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot3], confirmed [slot3]															x	x	x					x						askASlot
				x	x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot4], confirmed []				x																								explConf
	x	x	x																									presentVerbal
filled [slot4], confirmed [slot4]				x																								askASlot
	x	x	x																									presentVerbal
filled [slot1, slot2], confirmed []													x	x				x	x	x			x	x	x			implConf
filled [slot1, slot2], confirmed [slot1, slot2]																		x	x	x			x	x	x			askASlot
													x	x														presentMM
filled [slot1, slot2], confirmed [slot1]																		x	x	x			x	x	x			implConf
													x	x														presentMM
filled [slot1, slot2], confirmed [slot2]																		x	x	x			x	x	x			implConf
													x	x														presentMM

Learned State Action Mappings

state space																												system action
slot values	**no. DB hits**																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	18	20	25	26	34	40	54	56	92	142	175	438	
filled [slot1, slot3], confirmed []															x	x	x				x							implConf
					x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot1, slot3], confirmed [slot1, slot3]															x	x	x				x							askASlot
					x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot1, slot3], confirmed [slot1]															x	x	x				x							implConf
					x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot1, slot3], confirmed [slot3]															x	x	x				x							implConf
					x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot1, slot4], confirmed []			x	x																								implConf
	x	x																										presentVerbal
filled [slot1, slot4], confirmed [slot1, slot4]				x																								askASlot
	x	x	x																									presentVerbal
filled [slot1, slot4], confirmed [slot1]				x																								implConf
	x	x	x																									presentVerbal
filled [slot1, slot4], confirmed [slot4]				x																								implConf
	x	x	x																									presentVerbal
filled [slot2, slot3], confirmed []							x	x	x	x	x	x	x	x		x	x											implConf
filled [slot2, slot3], confirmed [slot2, slot3]																x	x											askASlot
				x	x		x	x	x	x	x	x	x	x														presentMM
filled [slot2, slot3], confirmed [slot2]																x	x											implConf
				x	x		x	x	x	x	x	x	x	x														presentMM
filled [slot2, slot3], confirmed [slot3]																x	x											implConf
				x	x		x	x	x	x	x	x	x	x														presentMM

Learned State Action Mappings

state space																												system action
slot values	**no. DB hits**																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	18	20	25	26	34	40	54	56	92	142	175	438	
filled [slot2, slot4], confirmed []		x	x	x																								explConf
	x																											presentVerbal
filled [slot2, slot4], confirmed [slot2, slot4]	x	x	x																									presentVerbal
filled [slot2, slot4], confirmed [slot2]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot2, slot4], confirmed [slot4]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot3, slot4], confirmed []	x																											presentVerbal
filled [slot3, slot4], confirmed [slot3, slot4]	x																											presentVerbal
filled [slot3, slot4], confirmed [slot3]	x																											presentVerbal
filled [slot3, slot4], confirmed [slot4]	x																											presentVerbal
filled [slot1, slot2, slot3], confirmed []															x	x	x											implConf
				x	x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot1, slot2, slot3], confirmed [slot1, slot2, slot3]															x	x	x											askASlot
				x	x	x	x	x	x	x	x	x	x	x	x	x	x											presentMM
filled [slot1, slot2, slot3], confirmed [slot1, slot2]															x	x	x											implConf
				x	x	x	x	x	x	x	x	x	x	x	x	x	x											presentMM
filled [slot1, slot2, slot3], confirmed [slot1, slot3]															x	x	x											implConf
				x	x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot1, slot2, slot3], confirmed [slot1]				x	x	x	x	x	x	x	x	x	x	x	x	x	x											implConf
filled [slot1, slot2, slot3], confirmed [slot2, slot3]															x	x	x											implConf
				x	x	x	x	x	x	x	x	x	x	x														presentMM
filled [slot1, slot2, slot3], confirmed [slot2]				x	x	x	x	x	x	x	x	x	x	x	x	x	x											implConf
filled [slot1, slot2, slot3], confirmed [slot3]				x	x	x	x	x	x	x	x	x	x	x	x	x	x											implConf

Learned State Action Mappings

state space																												system action
slot values	**no. DB hits**																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	18	20	25	26	34	40	54	56	92	142	175	438	
filled [slot1, slot2, slot4], confirmed []			x	x																								implConf
	x	x																										presentVerbal
filled [slot1, slot2, slot4], confirmed [slot1, slot2, slot4]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot1, slot2, slot4], confirmed [slot1, slot2]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot1, slot2, slot4], confirmed [slot1, slot4]				x																								presentMM
	x	x																										presentVerbal
filled [slot1, slot2, slot4], confirmed [slot1]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot1, slot2, slot4], confirmed [slot2, slot4]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot1, slot2, slot4], confirmed [slot2]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot1, slot2, slot4], confirmed [slot4]				x																								presentMM
	x	x	x																									presentVerbal
filled [slot2, slot3, slot4], confirmed []	x																											explConf
filled [slot2, slot3, slot4], confirmed [slot2, slot3, slot4]	x																											presentVerbal
filled [slot2, slot3, slot4], confirmed [slot2, slot3]	x																											presentVerbal
filled [slot2, slot3, slot4], confirmed [slot2, slot4]	x																											presentVerbal
filled [slot2, slot3, slot4], confirmed [slot2]	x																											presentVerbal
filled [slot2, slot3, slot4], confirmed [slot3, slot4]	x																											presentVerbal
filled [slot2, slot3, slot4], confirmed [slot3]	x																											presentVerbal
filled [slot2, slot3, slot4], confirmed [slot4]	x																											presentVerbal

Learned State Action Mappings

state space																									system action
slot values	**no. DB hits**																								
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	18	20	25	26	34	40	54	56	92	142	175	438	
filled [slot1, slot2, slot3, slot4], confirmed []	x																											explConf
filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot2, slot3]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot2, slot4]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot2]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot3, slot4]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot3]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot1, slot4]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot1]	x																											explConf
filled [slot1, slot2, slot3, slot4], confirmed [slot2, slot3, slot4]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot2, slot3]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot2, slot4]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot2]	x																											explConf
filled [slot1, slot2, slot3, slot4], confirmed [slot3, slot4]	x																											presentVerbal
filled [slot1, slot2, slot3, slot4], confirmed [slot3]	x																											explConf
filled [slot1, slot2, slot3, slot4], confirmed [slot4]	x																											explConf

References

Acomb K, Bloom J, Dayanidhi K, Hunter P, Krogh P, Levin E, Pieraccini R (2007) Technical support dialog systems: Issues, problems, and solutions. In: Proc. of the HLT/NAACL Workshop "Bridging the Gap: Academic and Industrial Research in Dialog Technologies"

Ai H, Litman D (2006) Comparing real-real, simulated-simulated, and simulated-real spoken dialogue corpora. In: Proc. of the AAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems, Boston, USA

Ai H, Litman D (2007) Knowledge consistent user simulations for dialog systems. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

Ai H, Raux A, Bohus D, Eskenazi M, Litman D (2007a) Comparing spoken dialog corpora collected with recruited subjects versus real users. In: Proc. of the 8th SIGdial workshop on Discourse and Dialogue

Ai H, Tetreault J, Litman D (2007b) Comparing user simulation models for dialog strategy learning. In: Proc. of the North American Meeting of the Association of Computational Linguistics (NAACL), Rochester, New York, USA, pp 1–4

Alexandersson J, Reithinger EMN (1995) A robust and efficient three-layered dialog component for a speech-to-speech translation system. In: Proc. of the 7th Conference of the European Chapter of the Association for Computational Linguistics (EACL)

Allen J, Ferguson G (2002) Human-machine collaborative planning. In: Proc. of the 3rd International NASA Workshop on Planning and Scheduling for Space

Allen J, Litman D (1987) A plan recognition model for subdialogues in conversations. Cognitive Science

Alpaydin E (2004) Introduction to Machine Learning. MIT Press

Androutsopoulos I, Ritchie G (2000) Database interfaces. In: Handbook of Natural Language Processing, Marcel Dekker Inc., pp 209–240

Araki M, Watanabe T, Doshita S (1997) Evaluating dialogue strategies for recovering from misunderstandings. In: Proc. of the IJCAI Workshop on Collaboration Cooperation and Conflict in Dialogue Systems

Asher N, Lascarides A, Lemon O (2011) Strategic conversation (STAC). website, `http://cordis.europa.eu/fetch?CALLER=FP7_PROJ_EN&ACTION=D&DOC=1&CAT=PROJ&RCN=99088` (11. September 2011)

Aust H, Oerder M, Seide F, Steinbiss V (1995) The Philips automatic train timetable information system. Speech Communication 17(3-4):249–262

Austin JL (1962) How to Do Things With Words. Oxford University Press

Balentine B, Morgan D (2001) How to Build a Speech Recognition Application (2nd edition). Enterprise Integration Group

Balogh J, Cohen M, Giangola JP (2004) Voice User Interface Design: Minimizing Cognitive Load. Addison Wesley Professional

Bangalore S, Rambow O, Walker M (2001) Natural language generation in dialog systems. In: Proc. of the 1st International Conference on Human Language Technology Research (HLT)

Bangalore S, Di Fabbrizio G, Stent A (2006) Learning the structure of task-driven Human-Human dialogs. In: Proc. of the 21st International Conference on Computational Linguistics (COLING) and 44th Annual Meeting of the Association for Computational Linguistics (ACL)

Banko M, Brill E (2001) Scaling to very very large corpora for natural language disambiguation. In: Porc. of the Annual Meeting of the Association for Computational Linguistics (ACL)

Barto A, Mahadevan S (2003) Recent advances in hierarchical Reinforcement Learning. Special Issue on Reinforcement Learning, Discrete Event Systems journal pp 41–77

Baxter J, Tridgell A, Weaver L (2001) Reinforcement Learning and chess. In: Machines that learn to play games, Nova Science Publishers, Inc., pp 91–116

Bazzi I, Glass J (2000) Modeling out-of-vocabulary words for robust speech recognition. In: Proc. of the International Conference of Spoken Language Processing (ICSLP)

Becker T, Lemon O, Young S, Blaylock N, Kruiff-Korbayova I, Georgilia K, Henderson J (2004) Proposed methods for multimodal experiments. Tech. rep., Deliverable 6.1, TALK Project

Becker T, Poller P, Schehl J, Blaylock N, Gerstenberger C, Kruijff-Korbayová I (2006) The SAMMIE system: Multimodal in-car dialogue. In: Proc. of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL)

Bellman R (1957) Dynamic Programming. Princeton University Press

Bennett C, Rudnicky A (2002) The carnegie mellon communicator corpus. In: Proc. of the International Conference of Spoken Language Processing (ICSLP)

Benyon D, Mival O (2007) Introducing the companions project: Intelligent, persistent, personalised interfaces to the internet. In: Proc. of the 21st British HCI Group Annual Conference (HCI)

Bernsen NO, Dybkjaer H, Dybkjaer L (1998) Designing Interactive Speech Systems: From First Ideas to User Testing. Springer Verlag

Bertomeu N, Uszkoreit H, Frank A, Krieger HU, Jörg B (2006) Contextual phenomena and thematic relations in database QA dialogues: results from a Wizard-of-Oz experiment. In: Proc. of the HLT-NAACL 2006 Workshop on Interactive Question Answering

Bishop CM (2006) Pattern Recognition and Machine Learning. Springer

Black AW, Burger S, Conkie A, Hastie H, Keizer S, Lemon O, Merigaud N, Parent G, Schubiner G, Thomson B, Williams JD, Yu K, Young S, Eskenazi M (2011) Spoken Dialog Challenge 2010: Comparison of Live and Control Test Results. In: Proc./ of SIGdial Workshop on Discourse and Dialogue

Blaylock N, Allen J (2005) A collaborative problem-solving model of dialogue. In: Proc. of the 6th SIGdial Workshop on Discourse and Dialogue

Blaylock N, Fromkorth B, Gerstenberger C, Kruijff-Korbayová I, Lemon O, Manchón P, Moos A, Rieser V, del Solar C, Weilhammer K (2006) Annotators handbook. Deliverable 6.2, TALK Project

Bohus D (2007) Error awareness and recovery in task-oriented spoken dialog systems. PhD thesis, Computer Science Department, Carnegie Mellon University

Bohus D, Rudnicky A (2002) Integrating multiple knowledge sources for utterance-level confidence annotation in the CMU Communicator spoken dialog system. Tech. rep., Technical Report CS-190, Carnegie Mellon University

Bohus D, Rudnicky A (2003) RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda. In: Proc. of the 8th European Conference on Speech Communication and Technology (Eurospeech)

Bohus D, Rudnicky A (2005a) Constructing accurate beliefs in spoken dialog systems. In: Poc. of the IEEE workshop on Automatic Speech Recognition and Understanding (ASRU)

Bohus D, Rudnicky A (2005b) A principled approach for rejection threshold optimization in spoken dialog systems. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

Bohus D, Langner B, Raux A, Black AW, Eskenazi M, Rudnicky A (2006) Online supervised learning of non-understanding recovery policies. In: Proc. of the IEEE/ACL workshop on Spoken Language Technology (SLT), Aruba,, pp 170–173

Boidin C, Rieser V, van der Plas L, Lemon O, Chevelu J (2009) Predicting how it sounds: Re-ranking alternative inputs to TTS using latent variables. In: Proc. of Interspeech/ICSLP, Special Session on Machine Learning for Adaptivity in Spoken Dialogue Systems

Bos J, Klein E, Lemon O, Oka T (2002) DIPPER: Description and formalisation of an Information-State Update dialogue system architecture. In: Proc. of the 4th SIGdial Workshop on Discourse and Dialogue

Bunt H (2007) Multifunctionality and multidimensional dialogue act annotation. In: Communication - Action - Meaning, A Festschrift to Jens Allwood, Gothenburg University Press, pp 237 – 259

Campbell M, Hoane AJ, Hsu FH (2002) Deep Blue. Artif Intell 134(1-2):57–83, DOI http://dx.doi.org/10.1016/S0004-3702(01)00129-1

Cao Y, Nijholt A (2008) Modality planning for preventing tunnel vision in crisis management. In: Proc. of the AISB Symp. on Multimodal Output Generation (MOG)

Carletta J (1996) Assessing agreement on classification tasks: the kappa statistic. Computational Linguistics 22(2):249–254

Carletta J, Isard A, Isard S, Kowtko JC, Doherty-Sneddon G, Anderson AH (1997) The reliability of a dialogue structure coding scheme. Computational Linguistics 23(1):13–31

Carletta J, Evert S, Heid U, Kilgour J (2005) The NITE XML Toolkit: data model and query. Language Resources and Evaluation Journal 39(42):313–334

Carpenter P, Jin C, Wilson D, Zhang R, Bohus D, Rudnicky AI (2001) Is this conversation on track? In: Proc. of the European Conference on Speech Communication and Technology (Eurospeech), Aalborg, Denmark, p 2121

Cheyer A, Martin DL (2001) The open agent architecture. Autonomous Agents and Multi-Agent Systems 40(1/2):143–148

Chickering D, Paek T (2007) Personalizing influence diagrams: Applying online learning strategies to dialogue management. User Modeling and User-Adapted Interaction 17(1-2):71–91

Chung G (2004a) Developing a flexible spoken dialog system using simulation. In: Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)

Chung G (2004b) Developing a flexible spoken dialog system using simulation. In: Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)

Clark H (1996) Using Language. Cambridge University Press

Clarkson P, Rosenfeld R (1997) Statistical Language Modeling Using the CMU-Cambridge Toolkit. In: Proc. of ESCA Eurospeech

Cohen J (1992) A power primer. Psychological Bulletin 112(1):155–159

Cohen WW (1995) Fast effective rule induction. In: Proceedings of the 12th International Conference on Machine Learning (ICML), Tahoe City, California, USA, pp 115–123

Cohn DA, Atlas L, Ladner RE (1994) Improving generalization with Active Learning. Machine Learning 15(2):201–221

Craggs R, McGee-Wood M (2005) Evaluating discourse and dialogue coding schemes. Computational Linguistics

Crook P, Lemon O (2010) Representing uncertainty about complex user goals in statistical dialogue systems. In: Proceedings of SIGDIAL

Crook P, Lemon O (2011a) Lossless value directed compression of complex user goal states for statistical spoken dialogue systems. In: Proceedings of Interspeech

Crook PA, Lemon O (2011b) Lossless Value Directed Compression of Complex User Goal States for Statistical Spoken Dialogue Systems. In: Proceedings of Interspeech

Cuayáhuitl H, Renals S, Lemon O, Shimodaira H (2005) Human-Computer Dialogue Simulation Using Hidden Markov Models. In: Proc. of the IEEE workshop on Automatic Speech Recognition and Understanding (ASRU), San Juan, Puerto Rico, pp 290–295

Cuayáhuitl H, Renals S, Lemon O, Shimodaira H (2006) Reinforcement Learning of Dialogue Strategies with Hierarchical Abstract Machines. In: Proc. of the IEEE/ACL workshop on Spoken Language Technology (SLT)

Dahlbäck N, Jönsson A, Ahrenberg L (1993) Wizard of Oz-studies – why and how. In: Proc. of the Workshop on Intelligent User Interfaces

van Deemter K (2009a) Utility and Language Generation: The case of Vagueness. Journal of Philosophical Logic pp 607–632

van Deemter K (2009b) What game theory can do for NLG: the case of vague language. In: 12th European Workshop on Natural Language Generation (ENLG)

Demberg V, Moore J (2006) Information presentation in spoken dialogue systems. In: Proc. of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)

Dempster A, Laird N, Rubin D (1977) Maximum likelihood from incomplete data via the EM algorithm. Journal of Royal Statistical Society B 39:1–38

Denecke M, Dohsaka K, Nakano M (2004) Fast Reinforcement Learning using stable function approximation. In: Proc. of the International Community of NLP (ICNLP)

Deng Y, Mahajan M, Acero A (2003) Estimating speech recognition error rate without acoustic test data. In: Proc. of the European Conference on Speech Communication and Technology (Eurospeech)

Dethlefs N, Cuayahuitl H (2010) Hierarchical Reinforcement Learning for Adaptive Text Generation. In: Proceedings of INLG

Dethlefs N, Cuayáhuitl H (2011) Hierarchical reinforcement learning and hidden markov models for task-oriented natural language generation. In: Proc. of 49th Annual Meeting of the Association for Computational Linguistics

Dietterrich T (2003) Machine learning. Nature Encyclopedia of Cognitive Science

Dix A, Finlay J, Abowd G, Beale R (1998) Human Computer Interaction. Prentice Hall

Doran C, Aberdeen J, Damianos L, Hirschman L (2001) Comparing several aspects of Human-Computer and Human-Human dialogues. In: Proc. of the 2nd SIGDIAL Workshop on Discourse and Dialogue

Dybkjaer L, Bernsen NO (2000) Usability evaluation in spoken language dialogue systems. In: Proc. of the ACL workshop on Evaluation for Language and Dialogue Systems

Eckert W, Levin E, Pieraccini R (1997) User modeling for spoken dialogue system evaluation. In: Proc. of the IEEE workshop on Automatic Speech Recognition and Understanding (ASRU), Santa Barbara, CA , USA, pp 80–87

Eckert W, Levin E, Pieraccini R (1998) Automatic evaluation of spoken dialogue systems. In: Proc. of Formal semantics and pragmatics of dialogue (TWLT13), Twente, NL, pp 99–110

Edlund J, Skantze G, Carlson R (2004) HIGGIS - a spoken dialogue system for investigating error handling techniques. In: Proc. of International Conference of Spoken Language Processing (Interspeech/ICSLP)

Engelbrecht KP, Möller S (2007) Pragmatic Usage of Linear Regression Models for the Predictions of User Judgements. In: Proc. of the 8th SIGdial workshop on Discourse and Dialogue, Antwerp, Belgium, pp 291–294

Engelbrecht KP, Quade M, Möller S (2009) Analysis of a new simulation approach to dialog system evaluation. Speech Communication 51(12):1234–1252

English M, Heeman P (2005) Learning mixed initiative dialog strategies by using Reinforcement Learning on both conversants. In: Proc. of the Human Language Technology Conference (HLT) and Conference on Empirical Methods in Natural Language Processing (EMNLP)

Eugenio BD, Glass M (2004) The kappa statistic: a second look. Computational Linguistics 30(1-2):95–101

Fabbrizio GD, Tur G, Hakkani-Tür D (2004) Bootstrapping spoken dialog systems with data reuse. In: Proc. of the 5th SIGdial Workshop on Discourse and Dialogue, Association for Computational Linguistics

Fabbrizio GD, Tur G, Hakkani-T-Tür D, Gilbert M, Renger B, Gibbon D, Liu Z, Shahraray B (2008) Bootstrapping spoken dialogue systems by exploiting reusable libraries. Journal of Natural Language Engineering

Field A (2005) Discovering statistics using SPSS (2nd Edition). Sage publications

Field A, Hole G (2003) How to design and report experiments. Sage publications

Filisko E, Seneff S (2005) Developing city name acquisition strategies in spoken dialogue systems via user simulation. In: Proc. of the 6th SIGdial Workshop on Discourse and Dialogue

Filisko E, Seneff S (2006) Learning decision models in spoken dialogue systems via user simulation. In: Proc. of the AAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems

Forlines C, Raj B, Schmidt-Nielsen B, Wittenburg K (2005) A comparison between spoken queries and menu-based interfaces for in-car digital music selection. In: Proc. of the International Conference on Human-Computer Interaction (INTERACT)

Foster ME, Oberlander J (2006) Data-driven generation of emphatic facial displays. In: Proc. of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Trento, Italy, pp 353–360

Frampton M (2008) Context models for dialogue strategy learning (to be submitted). PhD thesis, School of Informatics, University of Edinburgh

Frampton M, Lemon O (2006) Learning more effective dialogue strategies using limited dialogue move features. In: Proc. of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL)

Frampton M, Lemon O (2008) Using dialogue acts to learn better repair startegies for spoken dialogue systems. In: Proc. of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Frampton M, Lemon O (2009) Recent research advances in Reinforcement Learning in Spoken Dialogue Systems. Knowledge Engineering Review 24(4):375–408

Fraser NM, Gilbert GN (1991) Simulating speech systems. Computer Speech and Language 5:81–99

Frostad K (2003) Best practices in designing speech user interfaces, `http://msdn2.microsoft.com/en-us/library/ms994646.aspx` (3 jan 2008)

Gabsdil M, Lemon O (2004) Combining acoustic and pragmatic features to predict recognition performance in spoken dialogue systems. In: Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL), Barcelona, Spain, pp 343–350

Gasic M, Young S (2011) Effective handling of dialogue state in the hidden information state POMDP-based dialogue manager. ACM Transactions on Speech and Language Processing 7(3):4:1–4:28

Gasic M, Keizer S, Mairesse F, Schatzmann J, Thomson B, Young S (2008) Training and Evaluation of the HIS POMDP Dialogue System in Noise. In: Proc. of SIGdial Workshop on Discourse and Dialogue

Gasic M, Jurcicek F, Keizer S, Mairesse F, Thomson B, Yu K, Young S (2010) Gaussian processes for fast policy optimisation of a pomdp dialogue manager for a real-world task. In: Proceedings of SIGDIAL

Georgila K, Henderson J, Lemon O (2005a) Learning user simulations for information state update dialogue systems. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Lisbon, Portugal, pp 893–896

Georgila K, Lemon O, Henderson J (2005b) Automatic annotation of COMMUNICATOR dialogue data for learning dialogue strategies and user simulations. In: Proc. of the 9th SEMdial Workshop on on the Semantics and Pragmatics of Dialogues

Georgila K, Henderson J, Lemon O (2006a) User simulation for spoken dialogue systems: Learning and evaluation. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Pittsburgh, USA, pp 1065–1068

Georgila K, Henderson J, Lemon O (2006b) User simulation for spoken dialogue systems: Learning and evaluation. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Pittsburgh, USA

Georgila K, Lemon O, Henderson J, Moore J (2009) Automatic annotation of context and speech acts for dialogue corpora. Natural Language Engineering 15(3):315–353

Ghahramani Z (2004) Unsupervised learning. In: Bousquet O, Raetsch G, von Luxburg U (eds) Advanced Lectures on Machine Learning, Spinger Verlag

Ginzburg J (2008) Semantics and Conversation. CSLI Publications

Goddeau D, Pineau J (2000) Fast Reinforcement Learning of dialogue strategies. In: Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Griol D, Hurtado LF, Segarra E, Sanchis E (2008) A statistical approach to spoken dialog systems design and evaluation. Speech Communication pp 666–682

Griol D, Callejas Z, López-Cózar R (2010) Statistical dialog management methodologies for real applications. In: SIGDIAL Conference'10, pp 269–272

Grosz BJ, Sidner CL (1986) Attention, intentions and the structure of discourse. Computational Linguistics 12(3-4):175–204

Gruenstein A, Seneff S (2006) Context-sensitive language modeling for large sets of proper nouns in multimodal dialogue systems. In: Proc. of the IEEE/ACL 2006 Workshop on Spoken Language Technology

Gruenstein A, Seneff S (2007) Releasing a multimodal dialogue system into the wild: User support mechanisms. In: Proc. of the 8th SIGDIAL Workshop on Discourse and Dialogue

Guo H, Stent AJ (2005) Trainable adaptable multimedia presentation generation. In: Proc. of the 6th International Conference on Multimodal Interfaces (ICMI)

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. MLR Special Issue on Variable and Feature Selection 358:1157–1182

Hajdinjak M, Mihelic F (2006) The PARADISE evaluation framework: Issues and findings. Computational Linguistics 32(2):263–272

Hall M (2000) Correlation-based feature selection for discrete and numeric class machine learning. In: Proc. of the 17th International Conference on Machine Learning (ICML), San Francisco, CA, USA, pp 359 – 366

Hartikainen M, Salonen EP, Turunen M (2004) Subjective evaluation of spoken dialogue systems using SERVQUAL method. In: Proc. of the 8th International Conference on Spoken Language Processing (Interspeech/ICSLP)

Hassel L, Hagen E (2005) Adaptation of an automotive dialogue system to users' expertise. In: Proc. of the 7th SIGdial Workshop on Discourse and Dialogue

Heeman P (2007) Combining Reinforcement Learning with Information-State Update rules. In: Proc. of the North American Chapter of the Association for Computational Linguistics Annual Meeting (NAACL), Rochester, New York, USA, pp 268–275

Henderson J, Lemon O (2008) Mixture Model POMDPs for Efficient Handling of Uncertainty in Dialogue Management. In: Proc. of the Annual Meeting of the Association for Computational Linguistics

Henderson J, Lemon O, Georgila K (2005) Hybrid Reinforcement/Supervised Learning for Dialogue Policies from COMMUNICATOR data. In: Proc. of the IJCAI Dialogue Workshop

Henderson J, Lemon O, Georgila K (2008) Hybrid Reinforcement / Supervised Learning of Dialogue Policies from Fixed Datasets. Computational Linguistics 34(4):487 – 513

Herczeg M (1994) Software-Ergonomie: Grundlagen der Mensch-Computer-Kommunikation. Addison-Wesley

Hirschberg J, Litman D, Swerts M (2001) Identifying user corrections automatically in spoken dialogue systems. In: Proc. of the North American Meeting of the Association of Computational Linguistics (NAACL), Pittsburgh, Pennsylvania, USA, pp 1–8

Hof A, Hagen E, Huber A (2006) Adaptive help for speech dialogue systems based on learning and forgetting of speech commands. In: Proc. of the 7th SIGdial Workshop on Discourse and Dialogue

Hone K, Graham R (2000) Towards a tool for the subjective assessment of speech system interfaces (SASSI). Natural Language Engineering 6:303–387

Hosemann T (1840) Viertes Kapitel: Abenteuer des Freiherrn von Münchhausen im Kriege gegen die Türken. In: Gottfried August Bürger. Wunderbare Reisen zu Wasser und zu Lande, Feldzüge und lustige Abenteuer des Freiherrn von Münchhausen

Hu J, Winterboer A, Nass C, Moore JD, Illowsky R (2007) Context & usability testing: user-modeled information presentation in easy and difficult driving condition. In: Proc. of the Conference on Human Factors in Computing Systems (CHI), pp 1343–1346

Huang X, Acero A, Hon HW (2001) Spoken Language Processing: A Guide to Theory, Algorithm and System Development. Prentice Hall PTR

Janarthanam S, Lemon O (2008) User simulations for online adaptation and knowledge-alignment in troubleshooting dialogue systems. In: Proc. of the 12th

SEMdial Workshop on on the Semantics and Pragmatics of Dialogues, London, UK, pp 149–156

Janarthanam S, Lemon O (2009) A Two-tier User Simulation Model for Reinforcement Learning of Adaptive Referring Expression Generation Policies. In: Proc./ of the Annual SIGdial Meeting on Discourse and Dialogue

Janarthanam S, Lemon O (2010a) Adaptive referring expression generation in spoken dialogue systems: Evaluation with real users. In: Proceedings of SIGDIAL

Janarthanam S, Lemon O (2010b) Learning adaptive referring expression generation policies for spoken dialogue systems. In: Krahmer E, Theune M (eds) Empirical Methods in Natural Language Generation, Lecture Notes in Computer Science, vol 5980, Springer, Berlin / Heidelberg, pp 67–84

Janarthanam S, Lemon O (2010c) Learning to Adapt to Unknown Users: Referring Expression Generation in Spoken Dialogue Systems. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL), Uppsala, Sweden, pp 69–78

Janarthanam S, Hastie H, Lemon O, Liu X (2011) 'The day after the day after tomorrow? ' A machine learning approach to adaptive temporal expression generation: training and evaluation with real users. In: Proceedings of SIGDIAL

Johnson-Laird PN (1983) Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness. Cambridge University Press

Jokinen K, Hurtig T (2006) User expectations and real experience on a multimodal interactive system. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Pittsburgh, USA, pp 1049–1055

Jönsson A, Dahlbäck N (1988) Talking to a computer is not like talking to your best friend. In: Proc. of the Scandinavian Conference on Artificial Intelligence

Jung S, Lee C, Kim K, Jeong M, Lee GG (2009) Data-driven user simulation for automated evaluation of spoken dialog systems. Computer Speech & Language 23:479–509

Jurafsky D, Martin JH (2000) Speech and Language Processing. An Introduction to Natrural Language Processing, Computational Linguistics, and Speech Recognition. Prentice-Hall, New Jersey

Kaelbling L, Littman M, Cassandra A (1998) Planning and acting in partially observable stochastic domains. Artificial Intelligence 101:99–134

Koller A, Petrick R (2008) Experiences with planning for natural language generation. In: ICAPS

Korthauer A, Kruijff-Korbayová I, Blaylock TBN, Gerstenberger C, Kaisser M, Poller P, Rieser V, Schehl J, Lemon O, Georgila K, Henderson J, andCarmen Del Solar PM (2006) Final report on multimodal experiments Part II: Experiments for data collection and technology evaluation. Tech. rep., Deliverable 6.2, TALK Project

Krause J, Hitzenberger L (1992) Computer Talk. G. Olms

Kruijff-Korbayová I, Becker T, Blaylock N, Gerstenberger C, Kaisser M, Poller P, Schehl J, Rieser V (2005a) An experiment setup for collecting data for adaptive output planning in a multimodal dialogue system. In: Proc. of the of 10th European Workshop on Natural Language Generation (ENLG)

Kruijff-Korbayová I, Blaylock N, Gerstenberger C, Rieser V, Becker T, Kaisser M, Poller P, Schehl J (2005b) Presentation strategies for flexible multimodal interaction with a music player. In: Proc. of the 9th SEMdial Workshop on on the Semantics and Pragmatics of Dialogues

Kruijff-Korbayová I, Becker T, Blaylock N, Gerstenberger C, Kaisser M, Poller P, Rieser V, Schehl J (2006a) The SAMMIE corpus of multimodal dialogues with an MP3 player. In: Proc. of the 5th International Conference on Language Resources and Evaluation (LREC), Genoa, Italy, pp 2018–2023

Kruijff-Korbayová I, Rieser V, Becker T, Gerstenberger C (2006b) The Sammie Multimodal Dialogue Corpus Meets the Nite XML Toolkit. In: Proc. of the 5th Workshop on multi-dimensional Markup in Natural Language Processing

Kun A, Paek T, Medenica Z (2007) The effect of speech interface accuracy on driving performance. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

Lamel L, Minker W, Parboubek P (2000) Towards best practice in the development and evaluation of speech recognition components of a spoken language dialog system. Natural Language Engineering 6:305–322

Landragin F (2008) Pragmatics and human factors for intelligent multimedia presentation: A synthesis and a set of principles. In: Proc. of the AISB Symp. on Multimodal Output Generation (and)

Lane IR, Ueno S, Kawahara T (2004) Cooperative dialogue planning with user and situation models via example-based training. In: Proc. of the Workshop on Man-Machine Symbiotic Systems

Langley P, Arai S, Shapiro D (2004) Model-based learning with hierarchical relational skills. In: Proc. of the ICML Workshop on Relational Reinforcement Learning

Larson JA (2003) Commonsense guidelines for developing multimodal user interfaces, `www.larson-tech.com/MMGuide.html` (21 April 2008). URL `www.larson-tech.com/MMGuide.html`

Larsson S (2005) Dialogue systems: Simulations or interfaces? In: Proc. of the 9th SEMdial Workshop on on the Semantics and Pragmatics of Dialogues

Larsson S, Traum D (2000) Information state and dialogue management in the TRINDI dialogue move engine toolkit. Natural Language Engineering Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering 6(3-4):323–340

Lemon O (2006) Machine Learning and dialogue (course material). In: European Summer School in Logic, Language, and Information (ESSLLI), `http://homepages.inf.ed.ac.uk/olemon/lecture1.pdf` (21 April 2008)

Lemon O (2008) Adaptive natural language generation in dialogue using Reinforcement Learning. In: Proc. of the 12th SEMdial Workshop on on the Semantics and Pragmatics of Dialogues, London, UK

Lemon O (2011) Learning what to say and how to say it: joint optimization of spoken dialogue management and Natural Language Generation. Computer Speech and Language 25(2):210–221

Lemon O, Konstas I (2009) User simulations for context-sensitive speech recognition in spoken dialogue systems. In: European Conference of the Association for Computational Linguistics (EACL), Athens, Greece, pp 505–513

Lemon O, Liu X (2006) DUDE: a dialogue and understanding development environment, mapping business process models to Information State Update dialogue systems. In: Proc. of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)

Lemon O, Liu X (2007) Dialogue policy learning for combinations of noise and user simulations: transfer results. In: Proc. of the 8th SIGdial workshop on Discourse and Dialogue, pp 55–58

Lemon O, Pietquin O (2007) Machine Learning for spoken dialogue systems. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Antwerp, Belgium, pp 2685–2688

Lemon O, Bracy A, Gruenstein A, Peters S (2001) The WITAS multi-modal dialogue system I. In: Proc. of Eurospeech

Lemon O, Gruenstein A, Peters S (2002) Collaborative activities and multi-tasking in dialogue systems. Traitement Automatique des Langues (TAL), special issue on dialogue 43(2):131–154

Lemon O, Georgila K, Henderson J, Gabsdil M, Meza-Ruiz I, Young S (2005) Integration of learning and adaptivity with the ISU approach. Tech. Rep. TALK project deliverable D4.1, TALK Project

Lemon O, Georgila K, Henderson J (2006a) Evaluating Effectiveness and Portability of Reinforcement Learned Dialogue Strategies with real users: the TALK TownInfo Evaluation. In: Proc. of the IEEE/ACL workshop on Spoken Language Technology (SLT), Aruba, pp 170–173

Lemon O, Georgila K, Henderson J, Stuttle M (2006b) An ISU dialogue system exhibiting Reinforcement Learning of dialogue policies: Generic slot-filling in the TALK in-car system. In: Proc. of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)

Lemon O, Liu X, Shapiro D, Tollander C (2006c) Hierarchical Reinforcement Learning of dialogue policies in a development environment for dialogue systems: REALL-DUDE. In: Proc. of the 10th SEMdial Workshop on on the Semantics and Pragmatics of Dialogues, Potsdam, Germany, pp 185–186

Lemon O, Henderson J, Crook P, Liu X (2008) The q-mdp model. In: Proc. of the 22nd International Conference on Computational Linguistics (COLING)

Lemon O, Janarthanam S, Rieser V (2012) Reinforcement Learning Approaches to Natural Language Generation in Interactive Systems. In: Bangalore S, Stent A (eds) Natural Language Generation in Interactive Systems, Cambridge University Press

Levin E, Passonneau R (2006) A WOz variant with contrastive conditions. In: Proc. Dialog-on-Dialog Workshop, Interspeech, Pittsburgh, Pennsylvania, USA

Levin E, Pieraccini R (1997) A stochastic model of computer-human interaction for learning dialogue strategies. In: Proc. of the European Conference on Speech Communication and Technology (Eurospeech)

Levin E, Pieraccini R, Eckert W (1998) Using markov decision process for learning dialogue strategies,. In: Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Levin E, Pieraccini R, Eckert W (2000) A stochastic model of human-machine interaction for learning dialog strategies. IEEE Transactions on Speech and Audio Processing 8(1):11–23

Levinson SC (1983) Pragmatics. Cambridge University Press

Litman D (2002) Adding spoken dialogue to a text-based tutorial dialogue system. In: ITS Workshop on Empirical Methods for Tutorial Dialogue System

Litman D, Pan S (1999) Empiricallly evaluating an adaptable spoken dialogue system. In: Proc. of the 7th International Conference on User Modeling, Banff, Canada, pp 55–64

Litman DJ, Kearns MS, Singh S, Walke MA (2000) Automatic optimization of dialogue management. In: Proc. of the International Conference on Computational Linguistics (COLING)

Liu X, Rieser V, Lemon O (2009) A wizard-of-oz interface to study information presentation strategies for spoken dialogue systems. In: Proc. of the 1st International Workshop on Spoken Dialogue Systems

López-Cózar R, la Torre AD, Segura JC, Rubio AJ (2003) Assessment of dialogue systems by means of a new simulation technique. Speech Communication 40(3):387–407

MacKay D (2003) Information Theory, Inference, and Learning Algorithms. Cambridge University Press

Mattes S (2003) The lane-change-task as a tool for driver distraction evaluation. In: Strasser H, Kluth K, Rausch H, Bubb H (eds) Quality of Work and Products in Enterprises of the Future, ergonomia Verlag, Stuttgart, Germany, pp 57–60

McTear MF (2004) Towards the Conversational User Interface. Springer Verlag

Miller GA (1956) The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review 63:81–97

Möller S (2005a) Parameters for quantifying the interaction with spoken dialogue telephone services. In: Proc. of the 6th SIGdial Workshop on Discourse and Dialogue

Möller S (2005b) Towards generic quality prediction models for spoken dialogue systems - a case study. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

Möller S, Englert R, Engelbrecht K, Hafner V, Jameson A, Oulasvirta A, Raake A, Reithinger N (2006) MeMo: Towards automatic usability evaluation of spoken dialogue services by user error simulations. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Pittsburgh, Pennsylvania, USA, pp 1786–1789

Möller S, Smeele P, Boland H, Krebber J (2007) Evaluating spoken dialogue systems according to de-facto standards: A case study. Computer Speech & Language 21(1):26 – 53

Moore RK, Morris A (1992) Experiences collecting genuine spoken enquiries using woz techniques. In: Proc. 5th DARPA workshop on Speech and Natural Language

Mutschler H, Steffens F, Korthauer A (2007) Final report on multimodal experiments Part I: Evaluation of the SAMMIE system. Tech. rep., Deliverable 6.2, TALK Project

Nakatsu C (2008) Learning contrastive connectives in sentence realization ranking. In: Proc. of SIGdial Workshop on Discourse and Dialogue

Nakatsu C, White M (2006) Learning to say it well: Reranking realizations by predicted synthesis quality. In: Proc. of the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL)

Nass C, Barve S (2005) Wired for Speech: How Voice Activates and Advances the Human-Computer Relationship. MIT Press

Oh AH, Rudnicky AI (2002) Stochastic natural language generation for spoken dialog systems. Computer Speech & Language 16(3/4):387—407

Okamoto M, Yang Y, Ishida T (2001) Wizard of Oz method for learning dialog agents. Lecture Notes in Artificial Intelligence 2182:20–25

Osborne M, Baldridge J (2004) Ensemble-based Active Learning for parse selection. In: Proc. of the North American Meeting of the Association of Computational Linguistics (NAACL)

Oviatt S (2006) Human-centered design meets cognitive load theory: designing interfaces that help people think. In: Proc. of the 14th annual ACM international conference on Multimedia (MULTIMEDIA), ACM Press, New York, NY, USA

Oviatt S, Coulston R, Lunsford R (2004) When do we interact multimodally? Cognitive load and multimodal communication patterns. In: Proceedings of the 6th International Conference on Multimodal Interfaces (ICMI))

Paek T (2001) Empirical methods for evaluating dialog systems. In: Proc. ACL Workshop on Evaluation Methodologies for Language and Dialogue Systems, pp 3–10

Paek T (2006) Reinforcement Learning for spoken dialogue systems: Comparing strengths and weaknesses for practical deployment. In: Proc. Dialog-on-Dialog Workshop, Interspeech, Pittsburgh, Pennsylvania, USA

Paek T (2007) Toward evaluation that leads to best practices: Reconciling dialogue evaluation in research and industry. In: Proc. of the NAACL-HLT Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies, Rochester, New York, USA, pp 40–47

Paek T, Chickering D (2005) The Markov assumption in spoken dialogue management. In: Proc. of the 6th SIGDIAL Workshop on Discourse and Dialogue

Paek T, Horvitz E (1999) Uncertainty, utility, and misunderstanding: A decision-theoretic perspective on grounding in conversational systems. In: Proc. of the AAAI Fall Symposium on Psychological Models of Communication in Collaborative Systems

Paek T, Horvitz E (2000) Conversation as Action Under Uncertainty. In: Proc. of the 16th Conference on Uncertainty in Artificial Intelligence

Paek T, Horvitz E (2003) On the utility of decision-theoretic hidden subdialog. In: Proc. of the ISCA Workshop on Error Handling in Spoken Dialogue Systems

Paivio A (1990) Mental representations: A dual-coding approach. Oxford University Press, New York

Passonneau RJ, Epstein SL, Ligorio T, Gordon J (2011) Embedded wizardry. In: Proceedings of the SIGDIAL 2011 Conference, Association for Computational Linguistics, Portland, Oregon, pp 248–258, URL http://www.aclweb.org/anthology/W/W11/W11-2027

Pieraccini R, Huerta J (2005) Where do we go from here? Research and commercial spoken dialog systems. In: Proc. of the 6th SIGdial Workshop on Discourse and Dialogue

Pieraccini R, Suendermann D, Dayanidhi K, Liscombe J (2009) Are we there yet? research in commercial spoken dialog systems. In: Proceedings of TSD'09, pp 3–13

Pietquin O (2004) A framework for unsupervised learning of dialogue strategies. PhD thesis, Faculté Polytechnique de Mons

Pietquin O (2006) Machine Learning for spoken dialogue management: an experiment with speech-based database querying. In: Artificial Intelligence: Methodology,Systems& Applications, Springer Verlag

Pietquin O, Beaufort R (2005) Comparing ASR modeling methods for spoken dialogue simulation and optimal strategy learning. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

Pietquin O, Dutoit T (2006a) Dynamic bayesian networks for NLU simulation with application to dialog optimal strategy learning. In: Proc. of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)

Pietquin O, Dutoit T (2006b) A probabilistic framework for dialog simulation and optimal strategy learning. IEEE Transactions on Audio, Speech and Language Processing 14(2):589–599

Pietquin O, Hastie H (2011) A survey on metrics for the evaluation of user simulations. Knowledge Engineering Review URL http://www.metz.supelec.fr/metz/personnel/pietquin/pdf/KER_2011_OPHH.pdf, 15 pages - Accepted for Publication

Pietquin O, Renals S (2002) ASR system modeling for automatic evaluation and optimization of dialogue systems. In: Proc. of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1005671

Pietquin O, Geist M, Chandramohan S (2011a) Sample efficient on-line learning of optimal dialogue policies with kalman temporal differences. In: International Joint Conference on Artificial Intelligence (IJCAI 2011)

Pietquin O, Geist M, Chandramohan S, Frezza-Buet H (2011b) Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization. ACM Transactions on Speech and Language Processing 7(3), DOI 10.1145/1966407.1966412, URL http://www.metz.supelec.fr/metz/personnel/pietquin/pdf/ACM_TSLP_2011_OPMGSCHFB.pdf, 21 pages

Plüss B (2010) Non-cooperation in dialogue. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics Student Research Workshop, Association for Computational Linguistics, Stroudsburg, PA, USA, ACL-SRW 2010, pp 1–6

Polifroni J, Walker M (2006) Learning database content for spoken dialogue system design. In: Proc. of the IEEE/ACL workshop on Spoken Language Technology (SLT)

Polifroni J, Walker M (2008) Intensional Summaries as Cooperative Responses in Dialogue Automation and Evaluation. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)

Polifroni J, Chung G, Seneff S (2003) Generation of mixed-initiative dialogue systems from web content. In: Proc. of the European Conference on Speech Communication and Technology (Eurospeech)

Pon-Barry H, Clark B, Schultz K, Bratt E, Peters S (2004) Advantages of spoken language in dialogue-based tutoring systems. In: Proc. of the 7th International Conference on Intelligent Tutoring Systems

Potter B (1902) The Tale of Peter Rabbit. Frederick Warne & Co.

Prommer T, Holzapfel H, Waibel A (2006) Rapid simulation-driven Reinforcement Learning of multimodal dialog strategies in human-robot interaction. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Pittsburgh, Pennsylvania, USA, pp 1918–1924

Purver M, Ginzburg J, Healey P (2003a) On the means for clarification in dialogue. Current and New Directions in Discourse and Dialogue 22:235–255

Purver M, Healey P, King J, Ginzburg J, Mills G (2003b) Answering clarification questions. In: Proc. of the 4th SIGdial Workshop on Discourse and Dialogue

Putois G, Laroche R, Bretier P (2010) Enhanced monitoring tools and online dialogue optimisation merged into a new spoken dialogue system design experience. In: Proceedings of the SIGDIAL 2010 Conference, Association for Computational Linguistics, Tokyo, Japan, pp 185–192, URL `http://www.aclweb.org/anthology/W/W10/W10-4332`

Quarteroni S, Manandhar S (2008) Designing an open-domain Interactive Question Answering system. Natural Language Engineering (special issue on Interactive Question Answering, to appear)

Quinlan R (1993) C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco

Ranta A (2001) Grammatical framework. A type-theoretical grammar formalism. Journal of Functional Programming 14(2):145–189

Raspe RE (1785) The Surprising Adventures of Baron Munchhausen. Project Gutenberg

Reidsma D, Carletta J (2008) Reliability measurement: there's no safe limit. Computational Linguistics (to appear)

Rich C, Sidner C (1998) COLLAGEN: a collaboration manager for software interface agents. User Modeling and User-Adapted Interaction

Rieser V (2003) Entwurf und Implementierung einer Sprachschnittstelle für eine Waschmaschine ("Hermine, the talking wasching machine"). Master's thesis, Regensburg University

Rieser V (2008) Bootstrapping reinforcement learning-based dialogue strategies from wizard-of-oz data. PhD thesis, Saarbrueken Dissertations in Computational Linguistics and Language Technology, Vol 28

Rieser V, Lemon O (2006a) Cluster-based user simulations for learning dialogue strategies. In: Proc. of the 9th International Conference of Spoken Language Processing (Interspeech/ICSLP), Pittsburgh, Pennsylvania, USA, pp 1766–1772

Rieser V, Lemon O (2006b) Using logistic regression to initialise reinforcement-learning-based dialogue systems. In: Proc. of the IEEE/ACL workshop on Spoken Language Technology (SLT), Aruba, pp 190–193

Rieser V, Lemon O (2006c) Using Machine Learning to explore human multimodal clarification strategies. In: Proc. of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL), Sydney, Australia, pp 659–666

Rieser V, Lemon O (2007) Learning dialogue strategies for interactive database search. In: Proc. of the 10th International Conference of Spoken Language Processing (Interspeech/ICSLP), Special Session on Machine Learning for Spoken Dialogue Systems

Rieser V, Lemon O (2008a) Automatic learning and evaluation of user-centered objective functions for dialogue system optimisation. In: Proc. of the 6th International Conference on Language Resources and Evaluation (LREC), pp 2356–2361

Rieser V, Lemon O (2008b) Does this list contain what you were searching for? Learning adaptive dialogue strategies for interactive question answering. Natural Language Engineering 15(1):55–72

Rieser V, Lemon O (2008c) Learning effective multimodal dialogue strategies from Wizard-of-Oz data: Bootstrapping and evaluation. In: Proc. of the 21st International Conference on Computational Linguistics and 46th Annual Meeting of the Association for Computational Linguistics (ACL/HLT), Columbus, Ohio, USA, pp 638–646

Rieser V, Lemon O (2008d) Simulation-based learning of optimal multimodal presentation strategies from Wizard-of-Oz data. In: Proc. of the AISB Symp. on Multimodal Output Generation (MOG), Aberdeen, UK

Rieser V, Lemon O (2009a) Natural Language Generation as Planning Under Uncertainty for Spoken Dialogue Systems. In: Proc. of the Conference of European Chapter of the Association for Computational Linguistics (EACL), Athens, Greece, pp 683–691

Rieser V, Lemon O (2009b) Natural Language Generation as Planning Under Uncertainty for Spoken Dialogue Systems. In: Proc. of the Conference of European Chapter of the Association for Computational Linguistics (EACL)

Rieser V, Lemon O (2010a) Natural language generation as planning under uncertainty for spoken dialogue systems. In: Krahmer E, Theune M (eds) Empirical Methods in Natural Language Generation,, no. 5980 in Lecture Notes in Computer Science, Springer, Berlin / Heidelberg

Rieser V, Lemon O (2010b) Natural language generation as planning under uncertainty for spoken dialogue systems. In: Krahmer E, Theune M (eds) Empirical Methods in Natural Language Generation, Lecture Notes in Computer Science, vol 5980, Springer, Berlin / Heidelberg, pp 105–120

Rieser V, Lemon O (2011) Learning and evaluation of dialogue strategies for new applications: Empirical methods for optimization from small data sets. Computational Linguistics 37(1)
Rieser V, Moore J (2005) Implications for generating clarification requests in task-oriented dialogues. In: Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL), Ann Arbor, Michigan, USA, pp 239 –246
Rieser V, Kruijff-Korbayová I, Lemon O (2005) A corpus collection and annotation framework for learning multimodal clarification strategies. In: Proc. of the 6th SIGdial Workshop on Discourse and Dialogue, Lisbon, Portugal, pp 97–106
Rieser V, Lemon O, Liu X (2010) Optimising Information Presentation for Spoken Dialogue Systems. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Uppsala, Sweden, pp 1009–1018
Rieser V, Keizer S, Lemon O, Liu X (2011) Adaptive Information Presentation for Spoken Dialogue Systems: Evaluation with human subjects. In: Proceedings of ENLG
Robertson J, Cross B, Macleod H, Wiemer-Hastings P (2004) Children's interactions with animated agents in an intelligent tutoring system. Journal of Artificial Intelligence in Education 14(3,4):335–357
Rodriguez KJ, Schlangen D (2004) Form, intonation and function of clarification requests in German task-orientated spoken dialogues. In: Proc. of the 8th SEMdial Workshop on on the Semantics and Pragmatics of Dialogues, Barcelona, Spain, pp 101–108
Roy N, Pineau J, Thurn S (2000) Spoken dialogue management using probabilistic reasoning. In: Proc. of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)
Rudnicky A (2005) Multimodal dialogue systems. In: Spoken Multimodal Human-Computer Dialogue in Mobile Environments, Spinger, chap Issues in Multimodal Spoken Systems and Components, pp 3–11
Rummery GA, Niranjan M (1994) On-line Q-learning using connectionist systems. Tech. rep., Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department
Sacks H, Schegloff E, Jefferson G (1974) A simplest systematics for the organization of turn-taking for conversation. Language 50:696–735
Sadek M, Ferrieux A, Cozannet A, Bretier P, Panaget F, Simonin J (1996) Effective Human-Computer cooperative spoken dialogue: The AGS demonstrator. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)
Salmen A (2002) Multimodale menüausgabe im fahrzeug (”multimodal menu-based interaction in the vehicle”). PhD thesis, University of Regensburg
Schatzmann J (2008) Statistical user and error modelling for spoken dialogue systems. PhD thesis, Cambridge University Engineering Department
Schatzmann J, Georgila K, Young S (2005a) Quantitative evaluation of user simulation techniques for spoken dialogue systems. In: Proc. of the 6th SIGdial Workshop on Discourse and Dialogue, Lisbon, Portugal, pp 45–54

Schatzmann J, Stuttle M, Weilhammer K, Young S (2005b) Effects of the user model on simulation-based learning of dialogue startegies. In: Proc. of the IEEE workshop on Automatic Speech Recognition and Understanding (ASRU)

Schatzmann J, Weilhammer K, Stuttle M, Young S (2006) A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. Knowledge Engineering Review 21(2):97–126, DOI http://dx.doi.org/10.1017/S0269888906000944

Schatzmann J, Thomson B, Weilhammer K, Ye H, Young S (2007a) Agenda-based user simulation for bootstrapping a POMDP dialogue system. In: Proc. of the North American Meeting of the Association of Computational Linguistics (NAACL), Rochester, New York, USA, pp 149–152

Schatzmann J, Thomson B, Young S (2007b) Error simulation for training statistical dialogue systems. In: Proc. of the IEEE workshop on Automatic Speech Recognition and Understanding (ASRU), Kyoto, Japan, pp 526 –531

Schatzmann J, Thomson B, Young S (2007c) Statistical user simulation with a hidden agenda. In: Proc. of the 8th SIGdial Workshop on Discourse and Dialogue, Antwerp, Belgium, pp 273–282

Scheffler K, Young S (2001) Corpus-based dialogue simulation for automatic strategy learning and evaluation. In: Proc. NAACL Workshop on Adaptation in Dialogue Systems, Pittsburgh, Pennsylvania, USA, pp 64–70

Scheffler K, Young SJ (2002) Automatic learning of dialogue strategy using dialogue simulation and Reinforcement Learning. In: Proc. of the Human Language Technology Conference (HLT), pp 12–18

Schlangen D, Fernandez R (2007) Speaking through a noisy channel: Experiments on inducing clarification behaviour in human-human dialogue. In: Proc. of International Conference of Spoken Language Processing (Interspeech/ICSLP)

Schlangen D, Skantze G (2011) A general, abstract model of incremental dialogue processing. Dialogue and Discourse

Schulz S, Donker H (2006) An user-centered development of an intuitive dialog control for speech-controlled music selection in cars. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

Searle J (1969) Speech Acts. Cambridge University Press

Seung S, Opper M, Sompolinsky H (1992) Query by committee. Computational Learning Theory pp 287–294

Shapiro D (2001) Value-driven agents. PhD thesis, Stanford University, Department of Management Science and Engineering

Shapiro D, Langley P (2002) Separating skills from preference: Using learning to program by reward. In: Proc. of the 19th International Conference on Machine Learning (ICML), Sydney, Australia, pp 570–577

Shneiderman B (1997) Software Psychology: Human Factors in Computer and Information Systems, 3rd edn. Addison-Wesley

Singh S, Litman D, Kearns M, Walker M (2002) Optimizing dialogue management with Reinforcement Learning: Experiments with the NJFun system. JAIR 16:105–133

Skantze G (2003) Exploring human error handling strategies: implications for spoken dialogue systems. In: Proc. of the ISCA Tutorial and Research Workshop on Error Handling in Spoken Dialogue Systems

Skantze G (2005) Exploring human error recovery strategies: Implications for spoken dialogue systems. Speech Communication 43(3):325–341

Skantze G (2007a) Error handling in spoken dialogue systems. managing uncertainty, grounding and miscommunication. PhD thesis, Speech Communication, KTH Royal Institute of Technology, Sweden

Skantze G (2007b) Making grounding decisions: Data-driven estimation of dialogue costs and confidence thresholds. In: Proc. of the 8th SIGdial Workshop on Discourse and Dialogue, pp 206–210

Skantze G, Hjalmarsson A (2010) Towards incremental speech generation in dialogue systems. In: Proc./ of the Annual SIGdial Meeting on Discourse and Dialogue

Skantze G, Schlangen D (2009) Incremental dialogue processing in a micro-domain. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Association for Computational Linguistics, Stroudsburg, PA, USA, EACL '09, pp 745–753, URL `http://dl.acm.org/citation.cfm?id=1609067.1609150`

Sneed H, Merey A (1985) Automated software quality assurance. IEEE Transactions on Software Engineering 11(9)

Spitters M, Boni MD, Zavrel J, Bonnema R (2007) Learning to compose effective strategies from a library of dialogue components. In: Proc. of the 45th Annual Meeting of the Association for Computational Lingusitics (ACL)

Standage T (2002) The Turk: The Life and Times of the Famous Eighteenth-Century Chess-Playing Machine. New York: Walker

Steedman M, Petrick R (2007) Planning dialog actions. In: Proc. of the 8th SIGdial Workshop on Discourse and Dialogue

Stenchikova S, Stent A (2007) Measuring adaptation between dialogs. In: Proc. of the 8th SIGdial Workshop on Discourse and Dialogue

Stent A, Prasad R, Walker M (2004a) Trainable sentence planning for complex information presentation in spoken dialog systems. In: Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)

Stent A, Prasad R, Walker M (2004b) Trainable sentence planning for complex information presentation in spoken dialog systems. In: Association for Computational Linguistics

Stent A, Bangalore S, Fabbrizio GD (2008) Where do the words come from? learning models for word choice and ordering from spoken dialog corpora. In: Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Stuttle MN, Williams JD, Young S (2004) A framework for dialogue data collection with a simulated ASR channel. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP), Jeju, South Korea

Sutton R, Barto A (1998) Reinforcement Learning. MIT Press

Sweller J (1988) Cognitive load during problem solving: Effects on learning. Cognitive Science 12:257–285

Taylor P, Black A, Caley R (1998) The architecture of the Festival speech synthesis. In: Proc. of the 3rd International Workshop on Speech Synthesis

Tesauro G (1995) Temporal difference learning and TD-gammon. Commun ACM 38(3):58–68

Tetreault J, Litman D (2006) Using Reinforcement Learning to build a better model of dialogue state. In: Proc. of the 11th Conference of the European Association for Computational Linguistics (EACL)

Tetreault JR, Bohus D, Litman D (2007) Estimating the reliability of MDP policies: A confidence interval approach. In: Proc. of the North American Meeting of the Association of Computational Linguistics (NAACL)

Teufel S, van Halteren H (2004) Evaluating information content by factoid analysis: human annotation and stability. In: Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)

Thomson B, Young S (2010) Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. Computer Speech and Language 24(4):562–588

Thomson B, Schatzmann J, Weilhammer K, Ye H, Young S (2007) Training a real-world POMDP-based dialog system. In: Proc. of the HLT/NAACL Workshop “Bridging the Gap: Academic and Industrial Research in Dialog Technologies”

Thomson B, Schatzmann J, Young S (2008) Bayesian update of dialogue state for robust dialogue systems. In: Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Toney D (2007) Evolutionary Reinforcement Learning of spoken dialogue strategies. PhD thesis, School of Informatics, University of Edinburgh

Toney D, Moore JD, Lemon O (2006) Developing conversational interfaces with XCS. In: Proc. of the 9th International Workshop on Learning Classifier Systems (IWLCS)

Traum D, Swartout W, Gratch J, Marsella S (2008) A virtual human dialogue model for non-team interaction. Recent Trends in Discourse and Dialogue

Türck U (2001) The technical proceedings in SMARTCOM data collection: a case study. In: Proc. of the European Conference on Speech Communication and Technology (Eurospeech)

Ueno S, Lane IR, Kawahara T (2004) Example-based training of dialogue planning incorporating user and situation models. In: Proc. of the International Conference of Spoken Language Processing (ICSLP)

Varges S, Weng F, Pon-Barry H (2006) Interactive Question Answering and constraint relaxation in spoken dialogue systems. In: Proc. of the 7th SIGdial Workshop on Discourse and Dialogue

Wada F, Iwata M, Tano S (2001) Information presentation based on estimation of human multimodal cognitive load. IEEE pp 2924–2929

Walker M (2000) An application for Reinforcement Learning to dialogue strategiey selection in a spoken dialogue system for email. Artificial Intelligence Research 12:387–416

Walker M (2005) Can we talk? Methods for evaluation and training of spoken dialogue system. Language Resources and Evaluation 39(1):65–75

Walker M, Passoneau R (2001) DATE: A dialogue act tagging scheme for evaluation. In: Proc. of the Human Language Technology Conference (HLT)

Walker M, Litman D, Kamm C, Abella A (1997) PARADISE: a general framework for evaluating spoken dialogue agents. In: Proc. of the 35th Annual General Meeting of the Association for Computational Linguistics (ACL), Madrid, Spain, pp 271–280

Walker M, Fromer J, Narayanan S (1998a) Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In: Proc. of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (ACL/COLING), Montreal, Quebec, Canada, pp 780–78

Walker M, Litman D, Kamm C, Abella A (1998b) Evaluating spoken dialogue agents with PARADISE: Two case studies. Computer Speech and Language 12(3):1345–1351

Walker M, Kamm C, Litman D (2000) Towards developing general models of usability with PARADISE. Natural Language Engineering 6(3):363–377

Walker M, Passonneau R, Boland J (2001a) Quantitative and qualitative evaluation of DARPA Communicator spoken dialogue systems. In: Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)

Walker M, Rambow O, Rogati M (2001b) SPoT: A Trainable Sentence Planner . In: Proc. of the 2nd North American Meeting of the Association of Computational Linguistics (NAACL)

Walker M, Passonneau R, Aberdeen J, Boland J, Bratt E, Garofolo J, Hirschman L, Le A, Lee S, Narayanan S, Papineni K, Pellom B, Polifroni J, Potamianos A, Prabhu P, Rudnicky A, Seneff GSS, Stallard D, Whittaker S (2002a) Cross-Site Evaluation in DARPA Communicator: The June 2000 Data Collection. Submitted to Computer Speech and Language

Walker M, Rudnicky A, Prasad R, Aberdeen J, Bratt E, Garofolo J, Hastie H, Le A, Pellom B, Potamianos A, Passonneau R, Roukos S, Sanders G, Seneff S, Stallard D (2002b) DARPA Communicator: Cross-system results for the 2001 evaluation. In: Proc. of the 7th International Conference on Spoken Language Processing (Interspeech/ICSLP)

Walker M, Whittaker S, Stent A, Maloor P, Moore J, Johnston M, Vasireddy G (2004a) Generation and evaluation of user tailored responses in multimodal dialogue. Cognitive Science 28(5):811–840

Walker M, Whittaker S, Stent A, Maloor P, Moore J, Johnston M, Vasireddy G (2004b) User tailored generation in the MATCH multimodal dialogue system. Cognitive Science 28:81–840

Walker M, Stent A, Mairesse F, Prasad R (2007) Individual and domain adaptation in sentence planning for dialogue. Journal of Artificial Intelligence Research (JAIR) 30:413–456

Walker MA (1993) Informational redundancy and resource bounds in dialogue. PhD thesis, University of Pennsylvania

Wang H, Hamerich S, Hennecke M, Schubert V (2005) Speech-controlled media file selection on embedded systems. In: Proc. of the 6th SIGdial Workshop on Discourse and Dialogue

Watanabe T, Araki M, Doshita S (1998) Evaluating dialogue strategies under communication errors using computer-to-computer simulation. IEICE transactions on information and systems E81-D(9):1025–1033

Watkins C, Dayan P (1992) Q-learning. Machine Learning 8:279–292

Webb N, Webber B (eds) (2008) Journal of Natural Language Engineering: Special Issue on Interactive Question Answering, Cambridge University Press

Weber C, Elshaw M, Mayer NM (eds) (2008) Reinforcement Learning, Theory and Applications. I-Tech Education and Publishing

Weilhammer K, Stuttle M, Young S (2006) Bootstrapping language models for dialogue systems. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

Weinschenk S, Barker DT (2000) Designing Effective Speech Interfaces. Wiley

Whittaker S, Walker M, Moore J, Whittaker S, Walker M, Moore J (2002) Fish or fowl: A Wizard of Oz evaluation of dialogue strategies in the restaurant domain. In: Proc. of the International Conference on Language Resources and Evaluation (LREC)

Wickens C (2002) Multiple resources and performance prediction. Theoretical Issues in Ergonomic Science 3(2):159–177

Williams J (2006) Partially Observable Markov Decision Processes for Spoken Dialogue Management. PhD thesis, Cambridge University Engineering Department

Williams J (2007) A method for evaluating and comparing user simulations: The Cramer-von Mises Divergence. In: Proc. of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Kyoto, Japan, pp 508–513

Williams J (2008) The best of both worlds: Unifying conventional dialog systems and POMDPs. In: Proceedings of Interspeech

Williams J, Young S (2004a) Characterizing task-oriented dialog using a simulated ASR channel. In: Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Williams J, Young S (2004b) Using Wizard-of-Oz simulations to bootstrap Reinforcement-Learning-based dialog management systems. In: Proc. of the 4th SIGDIAL Workshop on Discourse and Dialogue, Sapporo, Japan, pp 135–139

Williams J, Young S (2007a) Partially Observable Markov Decision Processes for spoken dialog systems. Computer Speech and Language 21(2):231–422

Williams JD (2011) An empirical evaluation of a statistical dialog system in public use. In: Proc./ of the Annual SIGdial Meeting on Discourse and Dialogue

Williams JD, Young S (2007b) Scaling POMDPs for spoken dialog management. IEEE Trans on Audio, Speech, and Language Processing 15(7):2116–2129

Winterboer A, Hu J, Moore JD, Nass C (2007) The influence of user tailoring and cognitive load on user performance in spoken dialogue systems. In: Proc. of the 10th International Conference of Spoken Language Processing (Interspeech/ICSLP)

Witten IH, Frank E (2005) Data Mining: Practical Machine Learning Tools and Techniques (2nd Edition). Morgan Kaufmann, San Francisco

Wittgenstein L (1953) Philosophical Investigations. Blackwell Publishing

Xu W, Rudnicky A (2000) Task-based dialog management using an agenda. In: Proc. of the ANLP/NAACL Workshop on Conversational Systems

Young S (1995) Large vocabulary continious speech recognition: A review. In: Proc. of ASRU

Young S (2000) Probabilistic methods in spoken dialogue systems. Philosophical Trans Royal Society (Series A) 358(1769):1389–1402

Young S (2002) The statistical approach to the design of spoken dialogue systems. Tech. rep., Tech Report CUED/F-INFENG/TR.433, Cambridge University Engineering Department

Young S (2006) Using POMDPs for dialog management. In: Proc. of the IEEE/ACL Workshop on Spoken Language Technology (SLT)

Young S, Schatzmann J, Weilhammer K, Ye H (2007) The Hidden Information State Approach to Dialog Management. In: ICASSP 2007

Young S, Gasic M, Keizer S, Mairesse F, Schatzmann J, Thomson B, Yu K (2009) The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management. Computer Speech and Language 24(2):150–174

Zue V (2007) On organic interfaces. In: Proc. of the International Conference of Spoken Language Processing (Interspeech/ICSLP)

About the Authors

Dr Verena Rieser is a Lecturer in Computer Science at Heriot-Watt University, Edinburgh. She previously worked at Edinburgh University in the Schools of Informatics and GeoSciences, performing research in data-driven statistical methods for multimodal interfaces, as well as for modelling impacts of environmental change for sustainable development. She received her PhD (with distinction) from Saarland University in 2008, winning the Eduard-Martin prize.

Professor Oliver Lemon leads the Interaction Lab in the School of Mathematical and Computer Sciences (MACS) at Heriot-Watt University, Edinburgh. He previously worked at the School of Informatics, University of Edinburgh, and at Stanford University. His main expertise is in the area of machine learning methods for intelligent and adaptive multimodal interfaces, including work on Speech Recognition, Spoken Language Understanding, Dialogue Management, and Natural Language Generation. He applies this work in new interfaces for mobile search, virtual characters, Technology Enhanced Learning, and Human-Robot Interaction, in a variety of international research projects.

Please see `www.macs.hw.ac.uk/InteractionLab`.