

Modelling of Natural Dialogues in the Context of Speech-based Information and Control Systems



Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Markus M. Berg

Kiel
2014

1. Gutachter: Prof. Dr. rer. nat. habil.
Bernhard Thalheim
(Universität zu Kiel)
2. Gutachter: Prof. Michael F. McTear, PhD
(University of Ulster)
3. Gutachter: Ian M. O'Neill, PhD
(Queen's University Belfast)

Datum der mündlichen Prüfung: 24.07.2014

Diese Dissertation ist ebenfalls erschienen in der Reihe "Dissertationen zu Datenbanken und Informationssystemen" der Akademischen Verlagsgesellschaft AKA in Kooperation mit IOS Press. Veröffentlichung dieser Open Access Version der Universitätsbibliothek Kiel mit freundlicher Genehmigung des AKA Verlags.

This PhD Thesis has also been published in the series "Dissertations in Database and Information Systems" by Akademische Verlagsgesellschaft AKA in cooperation with IOS Press. Publication of this open access version by the University Library Kiel with kind permission of AKA.

Ordering information:

Modelling of Natural Dialogues in the Context of Speech-based Information and Control Systems

ISBN:

978-3-89838-508-4 (AKA)

978-1-61499-491-6 (IOS)

Publisher's website:

<http://www.aka-verlag.com>

<http://www.iospress.nl>

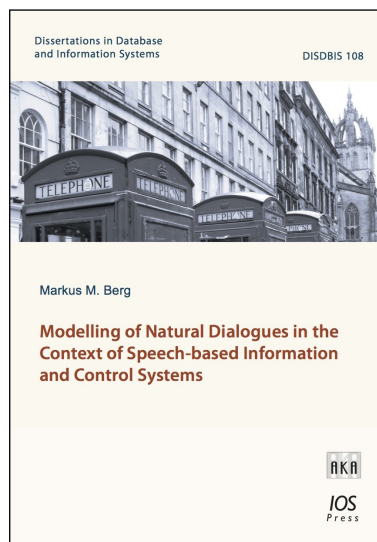
More information on the book:

<http://mmberg.net/go/aka>

<http://mmberg.net/go/ios>

Author's website:

<http://mmberg.net>



Abstract

Being able to communicate with machines by means of natural language is a long cherished dream of mankind. Consequently, it has been a central topic in many Science Fiction movies ever since. Apart from these fictional stories, today's speech dialogue systems often encounter criticism. In this dissertation, a model that addresses this criticism is developed in order to create more natural dialogue systems. This includes a study about the users' preferences and required features, the classification of utterances according to intention and answer type, the generation of system utterances, and the combination into a model that describes the characteristics and the information demand of the dialogue and that can be automatically processed by a dialogue engine. This research focusses on information and control systems, i.e. systems with clearly defined tasks, like travel booking or smart room control.

Initially, problems with today's dialogue systems are observed and features for a user-oriented and natural dialogue are identified. Based on this sociological evaluation, a dialogue model that addresses the results is built. The development of this model requires several steps.

First, the user utterance needs to be classified in relation to its intention. There are different possible formulations that all share the same goal. As simple keyword-based approaches have difficulties in spotting the difference between a request, a question or just the provision of new information, a machine learning approach is used to classify the utterances.

After having identified the user's aim, the actual proposition of the utterance can be interpreted more easily. Today's systems require the repeated definition of natural language understanding modules. Thus, in this thesis a taxonomy of relevant data types is developed. This allows us to connect natural language understanding methods to these types and contributes to a reusable question description that saves costs and facilitates the creation of natural dialogue systems.

Another issue that this thesis addresses, is the rigid formulation of system utterances, which makes systems boring and predictive. Instead, users wish the system to adapt to their preferred style of communication. Therefore, politeness and formality aspects are introduced into the question description model. Together with a language generation approach, this information is used to identify and realise a suitable formulation.

After having conducted the user study and having developed the classification and the generation process of utterances, the results are combined to produce a description of the whole dialogue, i.e. a set of questions, their possible answers and the intended action that has to be executed on the back-end. This description serves as input for the dialogue system and allows a fast and easy creation of natural dialogues by simply describing the questions and saving the work to create the dialogue logic and to implement the language understanding.

Finally, the resulting model is implemented to prove its functionality. We hope that this model contributes to improving human-computer-interaction and increasing user satisfaction.

Zusammenfassung

Die Kommunikation mit Maschinen unter Einsatz natürlicher Sprache ist schon seit langer Zeit ein Menschheitstraum und wurde daher schon vielfach in verschiedensten Science-Fiction-Filmen thematisiert. Abseits dieser Fiktion stoßen heutige Sprachdialogsysteme jedoch oft auf Kritik. In dieser Dissertation wird daher ein Modell entwickelt, das diese Kritik adressiert, um somit natürlichere Dialogsysteme erzeugen zu können. Dies beinhaltet eine Studie über die Vorlieben der Nutzer bei der Interaktion mit Dialogsystemen, die Klassifikation von Äußerungen hinsichtlich ihrer Aussageabsicht und des Antworttyps, die Generierung von Systemäußerungen, sowie die Kombination dieser Erkenntnisse zu einem Modell, das die Eigenschaften und den Informationsbedarf eines Dialoges beschreibt und automatisch durch ein Dialogsystem verarbeitet werden kann. Der Schwerpunkt liegt hierbei auf Informations- und Steuersystemen, d.h. auf Systemen mit klar definierten Aufgaben, wie z.B. der Reisebuchung oder Raumsteuerung.

Zunächst werden Probleme der Nutzer mit heutigen Dialogsystemen im Rahmen einer Umfrage und Nutzerstudie ermittelt und Eigenschaften für einen natürlichen Dialog identifiziert. Basierend auf dieser soziologischen Untersuchung wird in mehreren Schritten ein Dialogmodell entworfen.

Zu Beginn muss die Nutzeräußerung hinsichtlich ihrer Intention klassifiziert werden, da es verschiedene Formulierungsmöglichkeiten für die gleiche Zielstellung gibt. Bei der Anwendung einfacher schlüsselwortbasierter Ansätze bestehen Schwierigkeiten, den Unterschied zwischen einer Aufforderung, einer Frage oder der Bereitstellung neuer Informationen festzustellen. Daher kommt zur Klassifizierung ein maschinelles Lernverfahren zum Einsatz.

Nachdem das Ziel des Nutzers erkannt wurde, kann der eigentliche Inhalt der Aussage einfacher interpretiert werden. Heutige Systeme erfordern in verschiedenen Projekten die wiederholte Definition von Modulen zum Sprachverstehen. Daher wird in dieser Arbeit eine Taxonomie von relevanten Datentypen entworfen, die es erlaubt, die Module zum Sprachverstehen mit diesen Datentypen in Verbindung zu setzen und somit zu einer wiederverwendbaren Fragebeschreibung beiträgt. Dies hilft Kosten zu sparen und vereinfacht die Entwicklung von natürlichen Dialogsystemen.

Ein weiterer Aspekt der in dieser Arbeit adressiert wird, ist die starre Formulierung von Systemäußerungen, die Systeme uninteressant und vorhersehbar werden lässt. Stattdessen wünschen sich Nutzer ein System, das sich an deren bevorzugten Kommunikationsstil anpasst. Daher werden dem Modell die Parameter Höflichkeit und Formalität hinzugefügt. Zusammen mit einem Sprachgenerierungsverfahren werden diese Informationen genutzt, um eine passende Formulierung zu ermitteln und anschließend zu realisieren.

Nach Durchführung der Nutzerstudie und Klassifikation der Nutzeräußerungen, sowie der Beschreibung der Generierung von Systemäußerungen, werden die Ergebnisse zu einem Modell kombiniert, das den gesamten Dialog – d.h. eine Menge von Fragen, deren mögliche Antworten und die auszuführende Aktion – beschreibt. Diese Formalisierung dient als Eingabe für das Dialogsystem und ermöglicht die schnelle und einfache Erzeugung von natürlichen Dialogen, indem nur die Fragen beschrieben und Dialoglogik und Sprachverstehen dem Dialogsystem überlassen werden. Abschließend wird das entstandene Modell implementiert und dessen Funktionsfähigkeit nachgewiesen.

Contents

1	Introduction	10
1.1	Motivation	10
1.2	Previous Work	11
1.3	Problem Statement and Main Objectives	13
1.4	Contribution	14
1.5	Publications	15
1.6	Thesis Outline	18

I Towards the Processing of Natural Language Dialogues

2	A Linguistic View of Communication	20
2.1	Natural Language	20
2.2	Interaction and Communication	21
2.3	Dialogue and Conversation	23
2.4	Communication Models	24
2.4.1	Speech Acts, Austin and Searle	25
2.4.2	Four-Sides Model, Friedemann Schulz von Thun	28
2.4.3	The Cooperative Principle, Paul Grice	28
2.4.4	The Politeness Principle, Geoffrey Leech	30
2.4.5	Politeness Theory, Brown and Levinson	33
3	Computational Processing of Dialogues	36
3.1	Man-Machine-Communication	36
3.2	Dialogue Systems	37
3.2.1	Architecture and Components	38
3.2.2	Dialogue Strategy	39
3.2.3	Dialogue Control	41
3.2.4	Characteristics of Natural Dialogue Systems	44
3.3	Classification of Functions in Dialogues	50
3.3.1	Dialogue Acts	51
3.3.2	Interaction Patterns	54
3.3.3	Adjacency Pairs	55

4	Related Work	56
4.1	Studies towards the Interaction of Humans with Speech Interfaces	56
4.2	Modelling of Dialogues	59
4.3	Linguistic Style Adaptation	62
4.4	Modelling of Development Environments for Dialogue Systems	66
4.5	Conclusion	69

II About the User's Perspective on Natural Dialogue Systems

5	User Study about the Interaction with Speech Interfaces	72
5.1	Hypotheses	72
5.2	Setup and Methodology	74
5.3	Results	75
5.4	Conclusion	79
6	Survey on Spoken Dialogue Systems	80
6.1	Questions and Method	80
6.2	Results	80
6.2.1	Socio-Demographic	81
6.2.2	Experience and Satisfaction	82
6.2.3	Preferred Style (Self Assessment)	83
6.2.4	Preferred Style (Examples)	83
6.3	Analysis	84
6.3.1	Computer Usage	84
6.3.2	Gender	84
6.3.3	Age	85
6.3.4	Analysis of the Examples	86
6.3.5	Familiarity with Dialogue Systems	88
6.4	Conclusion	89
7	Conclusions	90

III Modelling of Information-seeking Dialogue Systems

8	Utterances and Their Effect on the System	93
8.1	Base Elements of a Dialogue	94
8.2	Back-end-oriented Dialogue Acts	95
8.2.1	Types of Back-end Functions	95
8.2.2	Selection of Dialogue Acts	96
8.3	Comparison with the CRUD-Approach	98
8.3.1	The CRUD Approach	98
8.3.2	CRUD in Natural Language Utterances	98

8.4	System-oriented Dialogue Acts	100
8.4.1	Elements of an Utterance	103
8.4.2	Description of Dialogue Utterances	104
8.4.3	Recognition of System-oriented Dialogue Acts	106
8.4.4	Application in a Dialogue System	109
8.5	Summary	112
9	Categorisation of Questions and Their Answers	113
9.1	Possibilities of Question Classification	113
9.2	Abstract Question Description	116
9.2.1	Type	117
9.2.2	Form	120
9.2.3	Context	121
9.3	Application of the AQD	123
9.4	Summary	125
10	Generation of System Interrogatives	126
10.1	Natural Language Generation	127
10.1.1	Combinatory Categorical Grammar	129
10.1.2	Question Generation	131
10.1.3	Linguistic Style	132
10.1.4	Telling Apart Politeness and Formality	132
10.2	Concept-to-Text	133
10.2.1	Definition of a Generation Grammar	135
10.2.2	Word Meaning Representation	138
10.3	Application in a Dialogue System	144
10.3.1	Programming Interface	146
10.3.2	Evaluation of Generated Example Interrogatives	149
10.4	Summary	150
11	Building a Dialogue Model	151
11.1	Dialogues and Tasks	151
11.2	Modelling of Information Transfer Objects	155
11.3	Actions and Follow-Up Questions	157
11.4	Resulting Dialogue Specification	157
11.5	Adaptivity	160
11.6	User Model	161
11.7	Summary	162
12	Dialogue System and Development Environment	163
12.1	User Interfaces and Communication with the Dialogue Engine	165
12.2	Runtime Dialogue Model	166
12.3	Dialogue Engine Architecture and Behaviour	167
12.3.1	Parsing and Natural Language Understanding	168

12.3.2	Actions	172
12.3.3	Utterance Templates	173
12.3.4	Task Stacking	174
12.4	Dialogue Development Environment	175
12.5	Summary	177

IV Scenarios and Evaluation

13	Use Cases	180
13.1	Room Control Scenario	180
13.2	Weather Information System	182
13.3	Trip Information System	183
13.4	Multiple Tasks: City Information System	184
13.5	Acting upon the Results: Number Guessing Game	185
14	Evaluation	187
14.1	Comparison with Voice XML	187
14.2	Comparison with AIML	188
14.3	Expert Interviews	189
14.4	Usability Test	191
15	Analysis and Discussion	195
15.1	Contributions	197
15.2	Limitations and Future Work	202
A	Appendix	204
A.1	Maximum Entropy Classifier Data	204
A.2	Politeness Scores for Different Question Types	205
A.3	Dialogue Definition	206
A.4	Dialogue Schema	212
A.5	NADIA Manual	215
A.5.1	Run Modes	215
A.5.2	Creating the Dialogue Model	217
A.5.3	Specifying the Dialogue Model with Java	222
A.6	Calculator Example	223
A.7	Expert Interviews	227
A.8	User Study	234
	Bibliography	238
	List of Figures	248
	List of Tables	250

The limits of my language mean
the limits of my world.

(Ludwig Wittgenstein, 1889 – 1951)

1

Introduction

Speech-based information systems offer dialogue based access to information via the phone. They avoid the complexity of computers/websites and introduce the possibility of accessing automated systems without being distracted from your visual focus (e.g. while driving a car) and without needing your hands or eyes (e.g. for visually impaired people or workers with gloves). Most people have already used spoken dialogue systems (SDS) in order to reserve tickets for the cinema, to check the credit of a pay-as-you-go phone or to look for the next bus. Generally, a SDS can be defined as a system that “enables a human user to access information and services that are available on a computer or over the Internet using spoken language as the medium of interaction” (Jokinen and McTear, 2009). By this means, these systems can offer a convenient way of retrieving information.

1.1 Motivation

However, Bringert (2008) identifies three major problems with current interactive speech applications:

1. They are not natural enough.
2. They are not usable enough.
3. They are not cheap enough.

These reasons are likely to be interdependent. Current dialogue systems are not usable enough because they are not natural enough because the development of natural dialogue systems is still very expensive and requires a lot of work. Furthermore, Pflieger (2008) claims that many research results have not been applied in industrial systems yet:

“Despite their commercial success, current commercial dialogue systems show only limited capabilities with regard to natural dialogue. Many of these systems do not support the resolution of referring or elliptical expressions and require reduced and simplified user commands. Moreover, true mixed-initiative dialogue management is still only available in research prototypes and users have to deal with system-driven dialogue management.”

He further argues that we need “*more powerful dialogue systems so that users do not need to adapt their natural conversational behavior to that of the dialogue system*”. Only this will allow us to realise “*the overall goal of employing a speech dialogue system [...] to ease the interaction with computers*”.

However, we are currently missing a generic dialogue model and a corresponding dialogue engine that can be reused in different projects without needing to start from scratch everytime. Also, a development environment, that allows us to create such systems, is missing. This is also confirmed by Turunen and Hakulinen (2000) who identify “*the lack of proper tools and frameworks*” as “*one of the main factors behind the difficulty of constructing advanced speech applications*”. Moreover, industry approaches like *VoixXML* are still very limited and render the creation and maintenance of natural dialogue systems difficult (Lemon et al., 2008), which has also been reported by Pflieger (2008):

“While VoiceXML-based dialogue systems typically consist of a set of well-defined VXML documents that are processed by a VXML-interpreter, state-of-the-art systems consist of a number of components of which each one typically comes with its own knowledge base. This means that the price for enhanced functionality and coverage of such systems is their general complexity and reduced maintainability.”

These observations have also been summarised by Glass and Weinstein (2001), who report:

“As anyone who has tried to create a mixed-initiative spoken dialogue system knows, building a system which interacts competently with users, while allowing them freedom in what they can say and when they can say it during a conversation, is a significant challenge. For this reason, many systems avoid this tactic, and instead take a more strategic approach which focuses on a directed dialogue. [...] Certainly, there are many technical difficulties to overcome, which include recognizing and understanding conversational speech, generating reasonable and natural responses, and managing a flexible dialogue strategy.”

On top of that, the demand for a generic, reusable dialogue definition has also been confirmed by some of our research projects, which we will briefly describe in the next section.

1.2 Previous Work

The topic of this thesis is based on the results and open issues of several research projects we have been conducting with our academic and industry partners over the last few years. We have observed that a generic dialogue development environment or supporting methodology for defining natural dialogues is missing. This leads to many different dialogue engines that are rebuild in every project although there are similarities between

the dialogues that would theoretically allow a reusable model. We now briefly summarise the goals of the afore-mentioned projects.

Electronic Ear (Version 2), 2009

The Electronic Ear is a simple speech interface for smart rooms that has been developed by BASIS GmbH Wismar and the University of Wismar. This application has been redesigned in my Master's Thesis (Berg, 2009) and now features an ontology to represent domain knowledge, includes dialogue features and is able to perform basic anaphora resolution. A graphical representation allows to simulate the dialogue system without actually having a smart room. The results contribute to the following project called MAIKE.

MAIKE, 2008–2010

The overall aim of the project “*Mobile Assistive Systems for Intelligent, Cooperative Rooms and Ensembles*” was the development of services that facilitate the interaction in work environments (e.g. conference rooms). Our part was the development of the speech processing unit of the multimodal user interface. For an overview of the results, please refer to the paper by Berg et al. (2010d).

Project partners: University of Wismar, University of Rostock, BASIS GmbH Wismar, brown-iposs GmbH.

Supported by the Ministry of Economy Mecklenburg-Vorpommern and the European Union (ESF, EFRE).

Travel Consult, 2008–2010

Current tourism information and booking portals offer a form-based interaction with a vast amount of settings that confuses many users. In this project, we have developed a text-based dialogue system, that allows the user to interact with the system by means of natural language. The results can be seen in the paper by Berg and Düsterhöft (2010).

Project partners: University of Wismar, MANET Marketing GmbH.

Supported by the Ministry of Economy Mecklenburg-Vorpommern and the European Union (ESF, EFRE).

Virtual Reception Desk, 2011–2013

In this project we are currently developing a mobile, speech-enabled app, where the user can get information about leisure activities and events around his location or travel destination. Moreover, the tourism provider can enter the latest news and offers over an IVR system to display them on his website.

Project partners: University of Wismar, MANET Marketing GmbH.

Supported by the Ministry of Economy Mecklenburg-Vorpommern and the European Union (ESF, EFRE).

1.3 Problem Statement and Main Objectives

It has been found that many current dialogue systems are not as usable as they could be. Many research outcomes are not being used because their integration is still very expensive due to a missing dialogue development environment that supports the design of natural dialogues. A basic requirement for such a software is a dialogue model that can be used to define the characteristics of the dialogue. That's why we develop such a model that allows a simpler creation of natural dialogue systems. By this means, we hope to contribute to more user friendly and more accepted dialogue systems that can be created in an easier manner than possible today. Before developing the model, we need to analyse the user behaviour and determine how users want to interact with speech-based dialogue systems. This leads to the following research questions and sub issues:

Q1: What problems exist in the interaction between man and machine and how do humans prefer to communicate with automated systems?

Q1.1: What features do current dialogue systems support?

Q1.2: What do users expect from dialogue systems?

Q1.3: What features make a dialogue natural?

Q2: How can a natural dialogue be described so that it is automatically processable by a dialogue engine?

Q2.1: How can we classify the intention of a user utterance and its effects on the system independently from the surface form?

Q2.2: How can the information demand of a dialogue be modelled?

Q2.3: How can we set the basis for a system that adapts its style to the user?

Q2.4: Does the resulting model allow the simple creation of natural dialogues?

In this connection we have to prove or falsify the following hypotheses:

T1: User utterances relate to different levels of the whole system. A differentiation in relation to the scope of an utterance is the basis to understand the utterance and to prevent misunderstandings.

T2: The same intention can be expressed by various formulations. A model that links these variations to the same meaning simplifies processing. However, we still need a detailed expressiveness in order to generate language. Hence, the model needs to support both views: interpretation and generation.

T3: Questions and answers together form a vital context and should be modelled and processed in one unit.

T4: Adaptive system utterances contribute to a higher user acceptance.

T5: The development of dialogue systems is still very complex today and needs to be simplified in order to be able to create natural dialogues.

Consequently, in this thesis we focus on the following aspects:

- Analysis of human communication with speech interfaces
- Classification and modelling of questions and answers as well as their effect on the back-end
- Language generation and linguistic style variation
- Development of a dialogue model that can be processed automatically and that can be created with the help of a dialogue development environment

1.4 Contribution

This thesis aims at improving and facilitating the design of dialogue systems and contributes to this research area on different levels. On a philosophical level, we have analysed dialogue system related terms like *natural language* or the difference between *interaction* and *communication* from different points of view and inferred reusable descriptions in order to form a common understanding. On a social level, we have identified how people like to interact with dialogue systems. We asked the participants what they dislike about current spoken dialogue systems and what features are missing or need improvement. Moreover, we have conducted a user study where the participants interact with a simulated dialogue system. While most of the participants prefer natural, i.e. human like dialogues, some of them also think that short, telegraphic utterances are more straightforward and effective. Hence, we need adaptable systems that can be configured to embody a certain style and behaviour. Furthermore, adaptive systems increase the naturalness even more because they are able to pursue the user's preferred style of interaction. Based on these findings, we have done research on a conceptual level that finally results in a dialogue model.

First, we have classified user utterances by their effect on the dialogue system. This led to *system-oriented dialogue acts* that allow us to prevent misunderstandings, e.g. if a user does not respond to a question but asks another one or if he wants to get the status of a device instead of changing it. In other words, these acts describe if the user wants to get information, provide information or execute an action. Afterwards, we have classified questions based on their intention, style and answer type. This resulted in an *abstract question description* that is used to specify the type of a system question. This description has then been extended with language generation features and serves as the input for our concept-to-text engine that generates system questions according to a specific style. The final question description contains three levels: *type*, *form* and *context* and is also used for natural language understanding as it defines the meaning and type of a question and allows to select the correct NLU module. We then analysed what different elements a dialogue consists of and how they can be described. This includes a combined approach where questions and their answers are modelled in one single unit in order to reflect the context. All of these findings gave rise to a dialogue model that can be used to specify dialogues and thus describe a whole dialogue system. This description serves

as the input for a dialogue engine. Through the separation of the dialogue’s information demand (dialogue description based on the dialogue model) from the actual dialogue behaviour (processing), we accomplish a reusable model of the dialogue that could be used by different dialogue engine implementations. So finally, on an applied level, we have developed a dialogue engine that is able to process our dialogue description and thus proven the viability of this approach. This system has been published on GitHub as NADIA (NAtural DIAlogue System).

In the final evaluation we have first presented some use cases that demonstrate the capabilities of the model and the developed system. Then, we have asked external experts to assess NADIA. They confirmed the demand for such a system, its usefulness and functionality and state that it contributes to an easier creation of natural dialogues. Afterwards, we have conducted a user study where non-expert users were asked to work through the manual and realise dialogues according to given tasks. This study was also successful and the users underlined the convenient development compared to existing approaches.

As you can see, this research contributes to the creation of a dialogue model that describes the information demand of a dialogue as well as the respective action that should be executed. This model includes approaches to make a dialogue more natural, e.g. language generation to automatically generate user-dependent formulations of system questions or the usage of dialogue acts to prevent misunderstandings, and introduces methods that facilitate dialogue development by providing an answer type hierarchy with associated language understanding modules, a set of back-end-connectors and the possibility to choose from different dialogue strategies, which all saves the developer a lot of work. This model has finally been implemented and can be interpreted by the dialogue engine. However, it should be noted, that this thesis does neither focus on the dialogue engine itself nor on the development of new dialogue strategies.

We don’t know any existing system that combines dialogue acts, concept-to-text language generation of system questions, answer types and natural language understanding modules into a dialogue model – with clearly separating the dialogue behaviour from the specification of the information demand – that can be processed by a generic and reusable dialogue engine. Consequently, we hope that this new approach contributes to more natural dialogue systems.

1.5 Publications

We now provide a list of articles that have been published throughout my¹ doctoral research. The following articles are directly connected to this dissertation and build the basis of some chapters of this thesis, as is indicated below.

1. Markus Berg. Natürlichsprachlichkeit in Dialogsystemen. *Informatik-Spektrum*, 36 (4):371–381, 2013. ISSN 0170-6012

¹In case of multiple authors I am the corresponding and main author.

- ▷ This journal paper discusses the current state of the art of dialogue systems and gives a comprehensive overview about this research area. This paper contributes to Section 3.2.4.
2. Markus M. Berg, Amy Isard, and Johanna D. Moore. An OpenCCG-Based Approach to Question Generation from Concepts. In Elisabeth Métais, Farid Meziane, Mohamad Saraee, Vijayan Sugumaran, and Sunil Vadera, editors, *Natural Language Processing and Information Systems*, volume 7934 of *Lecture Notes in Computer Science*, pages 38–52, Salford (UK), 2013b. Springer Berlin Heidelberg. ISBN 978-3-642-38823-1

▷ This article presents a concept-to-text approach to generate system questions in dialogue systems and serves as foundation for Chapter 10.

★ *This article has been selected among the top 8 papers (out of 30) of the NLDB 2013 conference*
 3. Markus Berg. Survey on Spoken Dialogue Systems: User Expectations regarding Style and Usability. In *XIV International PhD Workshop*, Wisla (Poland), 10 2012. ISBN 978-83-935427-0-3

▷ This article describes the results of a survey about how users like to interact with a dialogue system when only using speech. Chapter 6 is based on this publication.

★ *This article has been awarded with a Distinguished Paper Award.*
 4. Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Towards Interrogative Types in Task-oriented Dialogue Systems. In *17th International Conference on Applications of Natural Language Processing to Information Systems*, volume 7337 of *Lecture Notes in Computer Science*, Groningen (The Netherlands), 07 2012a. Springer. ISBN 978-3-642-31177-2

▷ This article introduces a model to describe the intention of questions independently from their surface forms and results in an early version of the *Abstract Question Description*, that is revised and extended in Chapter 9.
 5. Markus Berg. Towards the Modelling of Backend Functionalities in Task-Oriented Dialogue Systems. In *5th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, Poznan (Poland), 11 2011. ISBN 978-83-932640-1-8

▷ This paper gives an overview about *back-end-oriented dialogue acts* that build the foundation for the *system-oriented dialogue acts* that we describe in Chapter 8. These acts are important to correctly identify the desired effect on the dialogue system and the back-end.
 6. Markus Berg, Bernhard Thalheim, and Antje Düsterhöft. Dialog Acts from the Processing Perspective in Task Oriented Dialog Systems. In *Proceedings of the 15th*

Workshop on the Semantics and Pragmatics of Dialogue (SemDial 2011), pages 176–177, Los Angeles (USA), 09 2011b

▷ This publication gives a first sketch of *back-end-oriented dialogue acts* that are described in Chapter 8.

7. Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Adaptation in Speech Dialogues – Possibilities to Make Human-Computer-Interaction More Natural. In *Fifth Baltic Conference "Human-Computer-Interaction"*, Riga (Latvia), 08 2011a. ISBN 978-3-86009-127-2

▷ In this work, features are analysed that contribute to more usable and natural dialogue systems. The results are used in Section 3.2.4 and Section 11.5.

8. Markus Berg, Petra Gröber, and Martina Weicht. User Study: Talking to Computers. In *3rd Workshop on inclusive eLearning*, London (UK), 9 2010b. ISBN 978-3-88120-811-6

▷ The results of the user study being described in this paper, are the basis for Chapter 5. This study is conducted as a Wizard of Oz experiment and aims at analysing the preferred interaction style when having a simulated natural speech interface at hand.

The following articles complete my work in the context of speech dialogue systems but are not as extensively used to become the foundation of a chapter in this dissertation.

9. Markus Berg, Antje Düsterhöft, Nils Weber, and Christoph Eigenstetter. Wirtschaftlicher Mehrwert durch den Einsatz von Sprachtechnologien. In *Fachkongress Social Business – Tagungsband 2013*, pages 24–32. eBusiness-Lotse NordOst, 2013a. ISBN 978-3-00-043454-9
10. Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Query and Answer Forms for Sophisticated Database Interfaces. In *22nd European Japanese Conference on Information Modelling and Knowledge Bases*, Prague (Czech Republic), 07 2012b. ISBN 978-1-61499-176-2
11. Anna Biselli, Patrick Reipschläger, Markus Berg, and Antje Düsterhöft. Automatische Übersetzung von der Deutschen Gebärdensprache in die deutsche Lautsprache. In *6. Kongress Multimedialechnik*, Wismar (Germany), 10 2011
12. Markus Berg, Bernhard Thalheim, and Antje Düsterhöft. Integration of Dialogue Patterns into the Conceptual Model of Storyboard Design. In *Advances in Conceptual Modeling - Applications and Challenges, Proceedings of ER Workshops*, volume 6413 of *Lecture Notes in Computer Science*, pages 160–169, Vancouver (Canada), 11 2010c. Springer

13. Markus Berg, Nils Weber, Gernot Ruscher, and Sebastian Bader. Maike: Mobile Assistenzsysteme für intelligente kooperierende Räume und Ensembles. In *5. Kongress Multimediatechnik*, Wismar (Germany), 10 2010d
14. Petra Gröber, Martina Weicht, and Markus Berg. Inclusive eLearning - Special Needs and Special Solutions? In *3rd Workshop on inclusive eLearning*, London (UK), 9 2010
15. Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Integration of Natural Language Dialogues into the Conceptual Model of Storyboard Design. In *Natural Language Processing and Information Systems, 15th International Conference on Applications of Natural Language to Information Systems*, volume 6177 of *Lecture Notes in Computer Science*, pages 196–203, Cardiff (UK), 7 2010a. Springer
16. Markus Berg and Antje Düsterhöft. Website Interaction with Text-based Natural Language Dialog Systems. In *7. Wismarer Wirtschaftsinformatiktage*, Wismar (Germany), 6 2010. ISBN 978-3-939159-84-1
17. Markus Berg, Nils Weber, Christoph Eigenstetter, and Antje Düsterhöft. Natürlichsprachliche Dialoge in Raumsteuerungssystemen. In *4. Kongress Multimediatechnik*, Wismar (Germany), 10 2009
18. Nils Weber, Christoph Eigenstetter, Markus Berg, and Antje Düsterhöft. Ontologiespeicherung in Datenbanken im Kontext natürlichsprachlicher Dialogsysteme. In *21. Grundlagenworkshop von Datenbanken*, Rostock (Germany), 6 2009

1.6 Thesis Outline

This thesis is structured into four parts. The first part introduces important terms and theories in the context of computational linguistics and dialogue systems and then discusses related work.

In the second part we describe the results of a user study and a survey on the interaction of men with dialogue systems, which allows us to learn how humans prefer to interact with machines.

In the third part we present a dialogue model that considers the straightforward definition of natural dialogues. We first categorise natural utterances according to their intention and effect on the back-end which results in the definition of *system oriented dialogue acts*. Afterwards, we analyse questions and their respective answers based on their type and meaning. The result is an *abstract question description*. In the next step we describe how questions can be generated from concepts and develop a concept-to-text question generation programming interface. The results of those chapters contribute to a dialogue model that enables the dialogue designer to specify the behaviour and information demand of the dialogue. In the last chapter of this part, a dialogue engine that is able to process this model is developed.

The last part is dedicated to the evaluation of the proposed model and developed dialogue engine. Afterwards we conclude by summarising and discussing the results of this thesis.

Part I

Towards the Processing of Natural Language Dialogues

Synopsis

This Part of the thesis gives an introduction about relevant terms and theories in the areas of Linguistics and Dialogue Processing and serves as a prerequisite and foundation for the rest of the thesis. We start with the linguistic perspective in Chapter 2 and define terms that we use throughout this thesis, e.g. what we understand by natural language, what is the difference between interaction and communication and how we define a dialogue. We continue by giving an overview of different communication models, focussing on a functional description of utterances, maxims for effective communication, and politeness theories. In Chapter 3, we move on to the computational perspective and again explain basic terms like man-machine-communication or dialogue system, before we describe the architecture of dialogue systems and introduce different dialogue strategies. We then identify features of natural dialogue systems and introduce different possibilities to describe the function of utterances in dialogues. Afterwards, we analyse related work and describe the state of the art in Chapter 4. As this thesis deals with an interdisciplinary topic, we analyse studies about human interaction with speech interfaces, models about dialogue, adaptation of the linguistic style and the modelling of dialogue development environments.

2

A Linguistic View of Communication

This chapter gives an overview of (socio)linguistic, communication-related vocabulary, methods and theories that we need throughout this thesis. Most of the reviewed definitions originate from the philosophy of language and are therefore fuzzy and often describe personal views of single authors rather than being *universal* definitions. Therefore, we try to sharpen and extend these definitions from our point of view, i.e. from the perspective of dialogue systems, hoping that they are at least of universal use in the context of dialogue systems research. We then present relevant models on communication, the function of utterances and politeness.

2.1 Natural Language

The term *language* can be described as the performance of predefined, well-known and rule-based entities with a related meaning in order to communicate deliberately. This description is applicable for different types of languages. Apart from natural languages (including sign-languages) there also exist artificial/constructed languages like Esperanto or programming languages, as well as formal languages defined by a set of tokens and rules. Natural language can be described as “*a human written or spoken language as opposed to a computer language*” (Mifflin, 2009) whereas an artificial language is an “*invented language based on a set of prescribed rules and developed for a specific purpose, such as international communication or computer programming*” (Mifflin, 2009). In the context of this thesis we focus on natural languages. This includes both vocalised language (speech) and written language (text).

However, there are researchers who use the term ‘natural language’ in order to point out that their system is able to understand full sentences instead of only commands. So the definition of *natural* language can be viewed from two different perspectives. It could mean:

- every form of language that humans use to communicate (including single words and abstract commands) or
- the use of complex language in a human-like form using full sentences, following politeness rules and making use of colloquial words.

To illustrate the difference, imagine two systems which allow you to control a room by speech. One system is operated by saying “*light 4*” or “*light on*” while the other one enables you to say “*please turn on the light next to the window*”. Which one is natural? On the one hand, both are because both make use of words, which are part of a natural language and both communicate a message that can be understood by humans. On the other hand, only the second system uses a human-like formulation (i.e. like people would use to instruct other people).

But as men don’t always use full and grammatically correct sentences themselves without disqualifying these utterances as being not natural (imagine the single-word questions “*coffee?*” or “*lunch?*”, we follow the first approach, which says that every system that interprets utterances that can occur in human communication is a natural language processing system. The question if the system is able to process only single words or complete sentences is not a question of naturalness but a question of comprehensiveness or complexity of the system’s natural language understanding component. In this thesis we regard natural language as follows:

Convention 1 (*Natural Language*)

Natural language is the rule-based combination of human written or spoken words, that form meaningful expressions that people use in order to communicate.

Of course there are different degrees of expressiveness, length and complexity in natural language utterances.

2.2 Interaction and Communication

Interaction and *communication* are closely related and often used synonymously. Depending on the area of science, there are lots of different points of view, as you can see in the following descriptions for the term *communication*:

- The Oxford Dictionary describes communication as “*the imparting or exchanging of information by speaking, writing, or using some other medium*”
- In *information theory*, a communication system consists, according to (Shannon, 1948), of five parts: an information source that produces a message, a transmitter that operates on the message to make it a transmittable signal, a channel as the medium used to transmit the signal, a receiver that reconstructs the message from the signal, and a destination for whom the message is intended. Similarly, Schäfers (2006) defines communication as the “*exchange of information between a sender and a receiver with the help of specific signs and codes*”.
- In *action theory* communication is defined as a process in which individuals interrelate and agree on something (Schäfers, 2006).

- Mangold (1992), from the perspective of *computational linguistics*, defines communication as a natural-language-based mutual exchange of information between at least two partners.

Depending on the specific area, different assumptions are being made. In sociology, communication refers to human beings, linguistics requires the use of language, and computer science focusses on the encoding and transport of information. From a general point of view, communication can occur between persons, organisations, cultural groups, machines and the like. All definitions have in common that at least two interrelating parties are involved in the process.

Finding a universal but specific enough definition for all fields of science is nearly impossible and not necessarily useful. That's why we infer a definition for the scope of this thesis in the context of natural language processing.

Based on the meaning of the word, communication is sharing information, whereas interaction is a mutual influence. Abels and Stenger (1986) specify interaction as the "*mutual, meaningful, interrelated acting of persons*". Jaekel (1995) defines it as a "*process of mutual orientation of humans in certain situations*". Communication, on the other hand, "*takes place by interaction*" (Vester, 2008) and thus can be seen as a sub-process of it in which the communication partners try to agree on a particular meaning. This requires "*the recipient [to give] meaning to the utterances with the help of his own interpretational effort*" (Klann Delius, 2001). Consequently, communication is conscious behaviour. The definition of communication as being speech-based per se, like Mangold does, is problematic because of the existence of 'non-verbal communication'. That's why Klann Delius (2001) uses the term *verbal communication*.

We regard verbal communication as a special form of communication which is itself a form of interaction. So every form of communication is interaction, but not every form of interaction is communication. Throughout this thesis we work with the following conventions.

Convention 2 (*Interaction*)

Interaction is a mutual, meaningful, interrelated acting of individual actors.

This convention refers to Abels and Stenger (1986) but replaces persons with actors in order to include machines. Examples of interaction are (a) a driver who uses the steering wheel to influence the position of the car and at the same time perceives the change of position in order to control the steering process or (b) a person who changes the topic because his dialogue partner is sweating as a result of unease or stress or (c) a fish that hunts another fish.

Convention 3 (*Communication*)

Communication is a form of interaction that is characterised by a conscious sending of encoded information that represents the sender's intention and that needs to be decoded and interpreted by the receiver with the help of a common set of signs and rules in order to give it the correct meaning and infer the intended action.

The key difference between interaction and communication is that communication requires the understanding and reasoning over the given facts in order to infer an appropriate reaction. The interpretation of a message is the main source for misinterpretations because the set of rules for interpreting the message is not part of the message itself. A Polish speaker may say “tak” in order to express “yes”, whereas a Danish speaker would understand it as “thank you”. Also the nodding with the head can, depending on the culture, mean approval or denial.

Moreover, we assume that the sending of information is an active and conscious process. If a sender emits signals without intention (sweating, fluttering of eyelids), which is sometimes referred to as “unconscious communication”, we classify it only as interaction because sender and receiver cannot agree on a meaning if the sender is not aware that he has sent a message.

To distinguish interaction and communication, we give some examples. Pressing a switch is not communication because no message needs to be interpreted. Action and reaction are directly connected. Mouse gestures are communication because the gesture is a message that is actively sent and needs to be recognised and interpreted. A web search is communication as well, because the search for “film” can refer to a cinema (in which city?), an online-shop with movies, or a TV guide. In both cases, the word or gesture is a message that represents the sender’s intention and needs to be correctly interpreted by the receiver.

Convention 4 (*Verbal Communication*)

Verbal communication is communication by means of natural language.

As natural language includes written language, interacting with a chatbot is also verbal communication. This is sound because the bot needs to interpret the user’s message. In the context of this thesis, we always refer to verbal communication.

2.3 Dialogue and Conversation

A dialogue, generally speaking, is the exchange of information between two parties and is composed of the Greek words *dia* (through) and *logos* (speech, word). The main characteristic of a dialogue that delimits it from other forms of communication, is that a dialogue is always addressed communication, i.e. we know whom we are speaking to (a speech is communication, but it is not a dialogue). Moreover, a dialogue always involves natural language - may it be spoken or written.

Sometimes, also interacting with a graphical user interface (GUI) by means of a modal question window is called dialogue, because it is an addressed transfer of information between the user and one specific programme in the form of a question and an answer. Furthermore, user and programme mutually influence themselves constantly.

Stent (2001) defines a dialogue as “*interaction involving two participants, each of whom contributes by listening to the other and responding appropriately*” and Fong et al. (2002) describe it as “*the process of communication between two or more parties*”. It is “*a joint*

process” and “*requires sharing of information (data, symbols, context) and of control*”. Moreover, a dialogue has an aim and a central topic. In the context of this thesis, we regard dialogue as communication between agents (man or machine) and not between societies or cultures. As a subtype of verbal communication, it shares the same characteristics: it is a conscious, mutual, interrelated transfer of information between two partners encoded in natural language.

Convention 5 (*Dialogue*)

A dialogue is a form of verbal communication in which two participants exchange information about a common topic.

Dialogue and conversation are very similar. The Oxford Dictionary describes a conversation as “*a talk, especially an informal one, between two or more people*”. As an informal type of communication that can also occur among a group of people, conversation is not necessarily solution-oriented and somewhat more tentative. Moreover, a conversation can include changing topics. Searle (1992) underlines the missing aim by claiming that “*conversations [...] lack a particular purpose or point*”. However, Hagemann and Eckard (2001) disagree and refer to examination talks, trials or counselling interviews. In this thesis we define conversation as follows.

Convention 6 (*Conversation*)

A conversation is a form of verbal communication in which two or more participants exchange information about various topics in an informal way.

In speech technology, a conversational dialogue system describes a human-like system that is able to use informal language, adapts to the user and is able to perceive changes of the main topic of a talk (transition of domains). These systems may also include non-verbal information like gestures and emotions.

2.4 Communication Models

After having defined basic terms like communication and dialogue, we will now give an overview of some highly relevant work about the pragmatics of language, which we need in order to better understand the complexity of human communication and to be able to reuse parts of it for natural man-machine-communication. First, we introduce a *speech act* called system to describe utterances by the actions they represent. We continue by describing the different layers of messages and learn about the reasons for misinterpretations in human-human-communication. Afterwards, we present the *cooperative principle* that is necessary for effective communication. However, this model does not regard politeness, so finally we show two approaches that try to answer why we are polite, what makes an utterance polite and how we can measure politeness.

2.4.1 Speech Acts, Austin and Searle

What are we doing when saying “*Mind the gap*” and what do we mean by saying “*Could you open the window*”? And how are these utterances connected to actions like warning or requesting? Several philosophers gave thought to this question. Already Wittgenstein (1889 – 1951) initiated the reflection on the connection between words and acts when claiming: “*Worte sind Taten*”¹. Later, in 1962, also Austin philosophised about this topic and built the basis for Searle’s speech act theory from 1969. We now summarise the influential works of Austin and Searle.

How to Do Things with Words, Austin

The analysis of *performatives* marks the beginning of Austin’s posthumously published lectures about *speech acts* (Austin, 1975). In his considerations, he describes utterances that do not report or describe anything, nor are they true or false (i.e. they are no statements or claims). Instead, the *performing* of the utterance itself represents an act, e.g. “*I name this ship the Queen Elizabeth*” or “*I bet you six pence it will rain tomorrow*”. In other words, the act to name a ship is performed by saying the words “*I name*”, i.e. “*to say something is to do something*” or “*by saying or in saying something we are doing something*”. However, he remarks that betting can not only be achieved by uttering words, but also by using automated betting machines. Another interesting utterance is “*I promise...* ”. In this case, again, we don’t describe something, nor do we make a statement that can be true or false. However, we get into trouble if the promiser does not keep his promise. Austin calls this type of utterances neither true nor false, but *given in bad faith*. Also, a person who picks a player (by saying “*I pick you!*”) for his team out of a group of people who don’t want to play is a difficult situation. Again, we cannot say that the utterance is false (as in saying “*The house is red*” about a yellow house). Instead, Austin considers those utterances to be “*unhappy*”.

Later, Austin distinguishes between explicit (“*I order you to go!*”) and implicit performatives (“*Go!*”). Explicit performatives are utterances that clearly describe what act we perform: I order by saying “*I order*” or I bequeath by saying “*I bequeath*”. Implicit performatives, on the other hand, are not always unambiguous. The utterance “*There is a bull in the field*” “*may or may not be a warning, for I might just be describing the scenery*”.

After further problematic examples and difficulties in finding grammatical patterns for performatives, Austin decides in his seventh lecture to make a fresh start in order to find out what it means to *make* an utterance. First, he identifies three layers:

Phonetic act: the act to produce sounds (phones)

Phatic act: the act to produce words from a specific vocabulary

Rhetic act: the act to use the words and references appropriately

¹words are actions

Subsequently, Austin combines these three acts into a *locutionary act*, the act of saying something. By uttering the sentence “*Take a tea!*” we have correctly produced sounds that correspond to words and built a correct sentence. At this point, we don’t know if this utterance is a command, an advice, an offer or a request. So in the next step we interpret the *role* of a locution. This may be to ask, to answer, to inform, to describe or to warn and is called the *illocutionary act*. In the last step – the perlocutionary act – Austin describes the effect that the utterance has on the addressee, e.g. by asking a friend “*When are you going to the hairdresser’s again?*” just after she did, you may insult her. The following example describes the different layers by analysing the utterance “*Shoot her!*” (Austin, 1975):

Locutionary act: the act of saying something → He *said* to me “*Shoot her!*” meaning by *shoot* shoot and referring by *her* to her.

Illocutionary act: the act that is performed in saying something → He *urged* (or advised, ordered, etc.) me to shoot her.

Perlocutionary act: the act that is performed by saying something → He *persuaded* me to shoot her.

Austin concludes that by performing a locutionary act we also perform an illocutionary act, and by performing an illocutionary act, we perform a perlocutionary act. The essence of his work is the understanding that by uttering words we act and that this acting can be described on three different layers.

A Taxonomy of Illocutionary Acts, Searle

Searle, a student of Austin, further refines the speech act model. He starts with the question how to characterise the following utterances (Searle, 1969):

- “*Sam smokes habitually.*” → assertion
- “*Does Sam smoke habitually?*” → question
- “*Sam, smoke habitually!*” → order
- “*Would that Sam smoked habitually!*” → desire

All of these sentences obviously use words from the English language. But they are more than just a collection of words. In the first sentence, the speaker claims (or asserts) something, the second sentence is a question, the third one is an order and the last one is a desire. Additionally, by uttering any of these sentences, the speaker also performs reference and predication acts: he *refers* to a certain object *Sam* and he *predicates* the expression *smokes habitually*. Searle separates these acts from full speech acts like questioning or commanding because they can be the same in different speech acts.

He concludes, as well as Austin, that by uttering a sentence, the speaker performs acts on several layers. He then slightly adapts Austin’s speech act layers. Whereas in Austin’s

theory, the locutionary act consists of the phonetic, phatic and rhetic act, Searle renames it into *utterance act* and only includes the phonetic and phatic acts. Next, he extends the list by a new layer called *propositional act* that includes the earlier mentioned reference- and predication acts. The illocutionary and perlocutionary acts remain the same.

Later, Searle (1975) focusses on the illocutionary layer and identifies the following five categories of illocutionary acts:

Representatives “commit the speaker (in varying degrees) to something’s being the case, to the truth of the expressed proposition”, e.g. “The grass is green”. A representative can always be characterised as being true or false.

Directives “get the hearer to do something”, e.g. “I insist that you make your homework”

Commissives “commit the speaker (again in varying degrees) to some future course of action”, e.g. “I promise to do the dishes”

Expressives “express the psychological state”, e.g. “I apologise for being late”

Declarations refer to Austin’s performatives. The “successful performance of [a declaration] [...] guarantees that the propositional content corresponds to the world”, e.g. “You’re fired”

Apart from sentences where the speaker literally means what he says, there are far more complicated utterances where utterance meaning and sentence meaning come apart (Searle, 1979). By asking “Can you reach the salt?” the speaker performs not only a question, but primarily requests to pass the salt, i.e. “one illocutionary act is performed indirectly by way of performing another”. Searle calls this an *indirect speech act* and distinguishes between:

Primary illocutionary act: the indirect, meant act

Secondary illocutionary act: the direct, literal act

So in our last example, the primary act would be a request and the secondary act is a question. Another situation where we often encounter indirect speech acts is the case of rejections (Searle, 1979):

Student X: “Let’s go to the movies tonight.”

Student Y: “I have to study for an exam.”

When Y rejects X’s offer, he uses an indirect speech act. The literal meaning of Y’s utterance is just a statement. Finally, in his conclusions, Searle (1975) mentions that we must not mix illocutionary verbs with illocutionary acts because a verb can belong to different acts. I can “insist to go to the movies” and I can “insist that the answer is found on page 16”. The first utterance is a directive and the second one a representative. Hence, we are not able to assign verbs to specific acts.

2.4.2 Four-Sides Model, Friedemann Schulz von Thun

The utterance “*I tried to call you five times*” may be interpreted as anger (“*Do you think I have nothing better to do?*”), disappointment (“*Now it’s too late. I wanted to congratulate you on time!*”), worry (“*I thought something happened to you!*”) or curiosity (“*Where have you been all night?*”) and leaves a lot of room for misinterpretations. However, we are not able to further investigate why or how communication fails. According to von Thun (1983), who tries to address this issue, every message has four aspects or levels:

- *Matter* (fact-related): the matter the speaker informs about
- *Self-revealing* (speaker-related): what the speaker shows others about himself
- *Relationship* (listener-related): what the speaker thinks about the listener
- *Appeal* (intended effect-related): what the speaker wants the listener to do

This is referred to as the *four sides of a message*. Every message can be misinterpreted on every single layer, i.e. there are four dimensions on which communication can fail. Corresponding to these four aspects from the speaker’s point of view, the hearer interprets the message also on the same layers. Von Thun calls this the *Four-Ears-Model*. This model represents both speaker and listener. The following famous example by von Thun shows the different aspects of the model and the relation between what the speaker says and what the listener thinks he meant:

A man and his wife are having dinner. He sees capers and asks “*What’s the green in the soup?*”. His wife answers: “*If you don’t like it, you can cook yourself.*”

Speaker (four sides):

MATTER LAYER:	There is something green.
SELF-REVEALING LAYER:	I don’t know what it is.
RELATIONSHIP LAYER:	You should know what it is.
APPEAL LAYER:	Tell me what it is!

Listener (four ears):

MATTER LAYER:	There is something green.
SELF-REVEALING LAYER:	I don’t like it.
RELATIONSHIP LAYER:	You’re a bad cook.
APPEAL LAYER:	Only cook what I like in the future!

Depending on mood, relationship towards the speaker, situation and character, the hearer may unconsciously emphasise different layers of the message, i.e. he *hears* with different ears. This influences how he interprets the message and is the basis for misunderstandings.

2.4.3 The Cooperative Principle, Paul Grice

According to Grice (1989), every participant of a conversation will be expected to observe a *cooperative principle* that he formulates as:

“Make your conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged.”

Starting from here, Grice defines four categories of features for discourse that he calls *maxims of conversation* (Grice, 1989):

Maxim of Quantity: Make your contribution as informative as required (and don't make it more informative than is required).

Maxim of Quality: Try to make your contribution one that is true.

- Do not say what you believe to be false.
- Do not say that for which you lack adequate evidence.

Maxim of Relation: Be relevant.

Maxim of Manner: Be perspicuous.

- Avoid obscurity of expression.
- Avoid ambiguity.
- Be brief (avoid unnecessary prolixity).
- Be orderly.

While the first three maxims relate to *what* is said, the last one refers to *how* something is said. In this case, Grice refers to the purpose of a maximally effective exchange of information and he excludes aesthetic, social or moral maxims, such as *“be polite”*. Moreover, Grice argues that these maxims are not of equal importance, since saying something that is not true is more severe than being prolix.

There are three ways of failing to fulfil the maxims: A participant may *violate* a maxim (e.g. to mislead the hearer), he may *opt out* (*“I cannot say more; my lips are sealed”*), he may be *faced by a clash* (being unable to fulfil one maxim without violating another), e.g. if he needs to give more information but he has not enough evidence, or he may *flout* a maxim, i.e. he violates a maxim without wanting to disobey the cooperative principle and still expects the hearer to correctly interpret the message. An example of flouting is the question *“Do you want to come to my party?”* being answered with *“I have to work!”*. Here, it seems that the maxim of relation is disregarded. However, the speaker assumes that the hearer is able to interpret the answer correctly.

Grice believes that speakers follow the cooperative principle and that hearers assume that speakers follow it. This assumption explains the connection between the conversational maxims and what is called *conversational implicatures*. These implicatures can be implied by what has not been explicitly conveyed and by the application of the conversational maxims, as the following examples show:

- *“X is meeting a woman this evening.”*
→ The woman is not X's wife, sister or mother.

- “X has 20 €.”
→ X does not have more than 20 €.
- X: “In what city does Z live?” Y: “Somewhere in Scotland.”
→ Y does not know the city.
- X: “Do you want to go to the cinema tomorrow?” Y: “I will visit my parents.”
→ The visit is the reason that Y has no time and can’t make it to the cinema.

The third example refers to the maxim of quantity. We can imply that Y does not know the city, because we assume that he makes his contribution as informative as required. In the last example we clearly see the connection between the maxim of relation and the conversational implicature. Only because X assumes that Y’s answer is *relevant* he can conclude that this is the reason why Y cannot come and that his answer is not a random information.

2.4.4 The Politeness Principle, Geoffrey Leech

When analysing Grice’s cooperative principle (CP), Geoffrey Leech identifies two issues that it fails to answer (Leech, 1983): “*why people are often so indirect in conveying what they mean*” and “*what is the relation between sense and force when non-declarative types of sentence are being considered*”. He further reports about objections to Grice’s CP “*on the grounds that it does not stand up the evidence of real language use*”. Leech explains this with the fact that “*the majority of declarative sentences do not have an information-bearing function*”. Leech aims not at proposing a competitive model, but instead wants to solve these issues with the proposal of a supplement of Grice’s CP that he calls politeness principle, which “*can be seen [...] as a necessary complement*”. To explain the main problem with the CP, he gives the following example:

- A: “We’ll all miss Bill and Agatha, won’t we?”
B: “Well, we’ll all miss BILL.”

By deliberately not mentioning Agatha in B’s answer, he violates the maxim of quantity. We can derive that the group (or at least B) does not miss Agatha. Leech asks how we arrive at this implicature and states that CP is not the only basis because B could have added “*...but not Agatha*”. So “*B could have been more informative, but only at the cost of being more impolite*”. At this point, Leech formulates the first outcome of his politeness principle (PP):

- minimize the expression of negative beliefs
- maximize the expression of positive beliefs

Coming back to our example, we can see that B suppresses a negative belief, i.e. “*We won’t miss Agatha*”. Based on this insight, Leech formulates the connection between CP and PP as follows:

“The CP enables one participant in a conversation to communicate on the assumption that the other participant is being cooperative. In this the CP has the function of regulating what we say so that it contributes to some assumed illocutionary or discursal goal(s). It could be argued, however, that the PP has a higher regulative role than this: to maintain the social equilibrium and the friendly relations which enable us to assume that our interlocutors are being cooperative in the first place.”

In this way, the PP is not only a complement but a prerequisite. Leech now wants to quantify politeness in directive and commissive categories of illocutions. First, he places the propositional content on a cost-benefit-scale, as illustrated in Figure 2.1. As

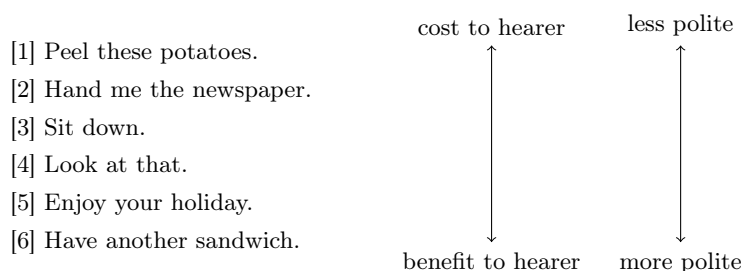


Figure 2.1: Cost-benefit scale (Leech, 1983)

you can see, the imperative mood is kept constant. However, the more benefit the addressee obtains, the more polite the request is being regarded (*“peel the potatoes”* vs *“have another sandwich”*). A second possibility *“of obtaining a scale of politeness is to keep the same propositional content [...] and to increase the degree of politeness by using a more and more indirect kind of illocution”*, as is illustrated in Figure 2.2. This leads

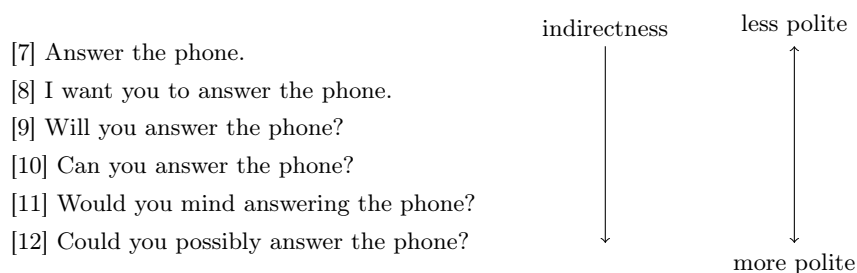


Figure 2.2: Indirectness scale (Leech, 1983)

to the preliminary conclusion that politeness is a function of indirectness. The reason why indirectness is perceived as politeness is that it biases *“the impositive more and more towards the negative choice, so that it becomes progressively easier for [the hearer] to say no”*, i.e. indirectness symbolises the degree to which it is allowed to not perform the intended action.

However, in another example Leech notices that two indirect illocutions (expressed as a negated question) have a completely different effect: “*Won’t you sit down?*” vs “*Can’t you sit down?*”. While the first utterance is seen as a polite offer, the second one is an impolite admonition. Another point is that a very indirect form can negatively influence politeness. While “*Have a sandwich!*” is a very positive way of making an offer, “*Would you mind having another sandwich?*” suggests some form of favour that you would do the addresser, which might lead to the conclusion that the sandwich is stale or poisoned. This shows how complex politeness – and human communication in general – is. However, going into more detail is beyond the scope of this introduction. Instead, we now summarise the results of Leech’s work, i.e. the maxims of the politeness principle (Leech, 1983):

1. TACT MAXIM (in impositives and commissives)
 - Minimize cost to *other*
 - Maximize benefit to *other*
2. GENEROSITY MAXIM (in impositives and commissives)
 - Minimize benefit to *self*
 - Maximize cost to *self*
3. APPROBATION MAXIM (in expressives and assertives)
 - Minimize dispraise of *other*
 - Maximize praise of *other*
4. MODESTY MAXIM (in expressives and assertives)
 - Minimize praise of *self*
 - Maximize dispraise of *self*
5. AGREEMENT MAXIM (in assertives)
 - Minimize disagreement between *self* and *other*
 - Maximize agreement between *self* and *other*
6. SYMPATHY MAXIM (in assertives)
 - Minimize antipathy between *self* and *other*
 - Maximize sympathy between *self* and *other*

The first two maxims refer to the cost-benefit scale and the next two, similarly, to the praise-dispraise scale. The last two maxims refer to the unipolar scales of agreement and sympathy. Like in Grice’s work as well, the importance of the maxims varies. Maxim 1 seems to be more important than 2, and 3 more important than 4. This is due to the fact that “*politeness is focused more strongly on other than on self*”. Also, every second sub-maxim seems to be less important than the first sub-maxim, which supports the hypothesis that negative politeness is more crucial than positive politeness; or in other words: it is more important not to say something impolite than being polite.

2.4.5 Politeness Theory, Brown and Levinson

Also Brown and Levinson (1987) analyse politeness strategies in human communication. Interestingly, their first work is from 1978, i.e. before Leech's politeness principle. In 1987 they published a revised version, where they comment on Leech's approach. Brown and Levinson also assume that Grice's theory of conversational implicature and his maxims are correct. But contrastingly to Leech, they don't think that the CP would make erroneous predictions without the PP, what they underline with three arguments:

- *“if we are permitted to invent a maxim for every regularity in language use, not only will we have an infinite number of maxims, but pragmatic theory will be too unconstrained to permit the recognition of any counter-examples”*
- *“the distribution of politeness (who has to be polite to whom) is socially controlled: it is not as if there were some basic modicum of politeness owed by each to all”*
- *“every discernable pattern of language use does not, eo ipso, require a maxim or principle to produce it”*

These arguments might be true in sociolinguistics. But from the modelling view it is beneficial to have guidelines on how politeness may be produced in order to use it in man-machine-interfaces.

Brown and Levinson do not regard the Gricean maxims as patterns in behaviour but instead refer to them as background assumptions. They argue that a partial answer to a question does not undermine the presumption of cooperation. Most seemingly uncooperative behaviour gets *“interpreted [...] at a deeper level”*, which makes it hard to undermine the CP. On the other hand, it is easy to undermine PP, i.e. by saying *“Shut your mouth”*. That's why Brown and Levinson don't accept the maxim-like status of the PP and are *“against setting up politeness principles as coordinate in nature to Grice's Cooperative Principle”*.

Instead, they propose their own model that we are going to present now. Brown and Levinson (1987) assume that *“all competent adult members of society have (and know each other to have) [face]”*. Face is *“the public self-image that every member wants to claim for himself”* and is related to the phrase *“to lose face”*. It can be divided into positive and negative face.

Positive face is the *“positive consistent self-image or personality claimed by interactants”*, i.e. the desire that the self-image is appreciated and approved of.

Negative face is the *“freedom of action and freedom from imposition”*, i.e. the claim to territories or rights to non-distraction.

The authors state that *“face is something that is emotionally invested, and that can be lost, maintained, or enhanced, and must be constantly attended to in interaction”*. They further conclude that people cooperate and maintain each others face because of the vulnerability of their own face. Thus, Brown and Levinson regard face as *wants*; the

want that your actions are unimpeded by others and the want that your wants are desirable to at least some others.

However, sometimes the face is at risk. Brown and Levinson call this *face-threatening acts* (FTA). Examples include commands, advices, compliments and threats because they aim at influencing H's (hearer) actions and thus restrict his freedom of action, or in other words: they threaten H's negative face. But also the positive face can be threatened. Insults, protests or critique attack H's desire for appreciation and approval. Apart from threats to H, S (speaker) can also threaten himself. By apologising, confessing or by accepting a compliment, S's positive face is at risk. In the latter case, S could threaten his positive face, because H may assume that S is arrogant (H: "*We would be lost without you*" - S: "*I know*"). Instead, a polite strategy would be to play down the compliment: "*Oh no, it was only possible with my great team!*". On the other hand, this utterance could also negatively influence S's positive face because H may think that S actually did not do the work. Examples for threatening S's negative face are thanks (S accepts his debt), excuses (an excuse of S may "*constitute in turn a criticism of H*"), or unwilling promises ("*S commits himself to some future action although he doesn't want to*"). This leads to different politeness strategies to save H's face when expressing an FTA is inevitable.

Bald On-Record: Do not minimize threats to H's face ("*Pass me your milk!*")

Positive Politeness: Minimize threats to H's positive face ("*Hey dude, is it OK if I take your milk?*")

Negative Politeness: Minimize threats to H's negative face ("*Excuse me, I was wondering if it is possible to get some of your milk?*")

Off-Record Indirectly describe your desire instead of asking H ("*Damn, I forgot to buy milk!*")

Bald on-record strategies are "*direct, clear, unambiguous and concise*" and extremely threaten H's negative face, so they should only be used in emergencies ("*Pass me the life belt!*") or if H is higher of rank (military, superior). Positive and negative politeness refers to the corresponding faces. Since the positive face is connected to appreciation, social closeness is emphasised. Moreover, S expresses that he belongs to the same social group and shares H's interests. Negative face relates to the freedom of actions, i.e. by reducing the force of the request, the face threat can be minimised. Hereby, S tries to assure H that he respects his negative-face desires. Connected with these strategies are three wants:

1. "*the want to communicate the content of the FTA*",
2. "*the want to be efficient or urgent*",
3. "*the want to maintain H's face to any degree*".

Only if 2. is greater than 3., S will not try to minimize the threat of H's face. Finally, Brown and Levinson describe three factors that influence the decision which politeness strategy to choose:

- the *social distance* $D(S,H)$ (symmetric),
- the *relative power* $P(S,H)$ & $P(H,S)$ (asymmetric),
- and the *absolute ranking* R of impositions in the particular culture.

If social distance is small, we may choose a positive politeness strategy and make use of in-group words like *dude*, *mum* or *honey*. If social distance is huge, a negative politeness strategy is more appropriate. But also the relative power has a big influence on the choice of politeness strategies. A teacher may use a bald on-record strategy towards his students (“*Silence!*”), whereas he would better use a positive politeness strategy when talking to his colleagues (“*I really like this song, but can you turn down the volume? I need to check some tests.*”). Finally, the absolute ranking of impositions refers to the threat level of the act: a request for a big favour requires a politer strategy than asking for a pen.

If language had been the creation not of poetry but of logic,
we should only have one.

(Friedrich Hebbel, 1813 – 1863)

3

Computational Processing of Dialogues

In the last chapter we gave an overview of terms and theories from a linguistic and philosophical perspective. In this chapter we introduce relevant terms and theories of man-machine-communication and dialogue systems.

3.1 Man-Machine-Communication

Man-machine-communication is a generic term for a number of user-interface related areas of research. Also speech interfaces and spoken dialogue systems are part of it. This area of research is known under several slightly differing terms, like man-machine-interfaces, man-machine-interaction, human-computer-interaction and man-machine-communication. We prefer the term *man-machine-communication* because we want to emphasise that:

- we don't necessarily need to use a computer directly (e.g. with mouse and keyboard), but instead we can also operate these systems over the phone. We are aware that a computer is always part of the processing, although not necessarily being the front-end.
- we use language in order to interact with the machine (i.e. referring to Convention 4, we could say *verbal* man-machine-communication).

Communicating with machines often leads to difficulties caused by an inconsistent design or a discrepancy between the users' expectations and the systems' capabilities. One reason is the way in which users regard the system. Edlund et al. (2006) propose that "*users may, knowingly or subconsciously, interpret the events that occur when interacting with spoken dialogue systems in more than one way*". They describe two different metaphors that relate to the different attitudes people have of the system: "*In the interface metaphor, the spoken dialogue system is perceived as a machine interface*" whereas "*in the human metaphor, the computer is perceived as a human-like creature*". They underline the plausibility of this theory by stating that most of today's SDS just offer an alternative to existing (machine) interfaces. On the other hand, also the human metaphor is plausible because speech is associated with human-likeness. Depending on how the user

perceives the system, he will make use of different formulations and dialogue strategies. When the system's and the user's metaphors do not correspond, this can result in the following non-ideal dialogues:

S: *"Tell me where you want to start!"*
U: *"Hi. I'd like to set off from London."*
S: *"I did not understand you. Please say the city where you want to start from!"*

S: *"How can I help you?"*
U: *"London"*
S: *"What do you want to do in London? I can offer accommodations and tickets for the cinema."*

The first dialogue system clearly follows the interface metaphor, while the user follows the human metaphor. The system expects the user to say a single word (like the response you would type into a web form), whereas the user answers in a full sentence that the system is not capable to understand. In the second example, the systems expects the user to pursue a dialogue based on the human metaphor, but the user answers by means of the machine metaphor, which results in an underspecified answer and forces the system to ask a clarification question.

In order to further characterise and analyse the underlying systems and connected problems, we first need to introduce basic terms in the next sections.

3.2 Dialogue Systems

Most people already came into contact with dialogue systems, mostly in form of automated call centres. These systems offer telephone-based, automated services like pre-qualification (determine the correct department for the customer's problem), bus timetable information, or topping up your pay-as-you-go phone. While some of these commercial systems still only use a key-based navigation (DTMF), others also support speech recognition. The first text-based dialogue systems were *ELIZA* (Weizenbaum, 1966) and *SHRDLU* (Winograd, 1972). While *ELIZA* was able to simulate a psychologist by reformulating the user's answers, with *SHRDLU* you could move blocks in a simulated world by natural language commands.

When speaking about dialogue systems in general, we include speech-based dialogues as well as text-based dialogues like chats, because we have defined a dialogue as a form of verbal communication.

Convention 7 (*Dialogue System*)

A dialogue system (DS) is a machine that verbally communicates with a human user to exchange information about a common topic.

This general convention not only includes passive services that users can access to get information, but potentially also active, intelligent agents that want to get information

from the user. However, when we speak of dialogue systems, we mostly refer to *task-oriented dialogue systems*, i.e. systems that are used for “*getting information, issuing instructions or providing a service*” (McTear, 2004). In these cases, the system has a clearly defined task that it is able to fulfil. Jokinen and McTear (2009), in a broader sense, give the following description:

“Task-oriented systems involve the use of dialogues to accomplish a task, which may be a fairly simple and well-defined task such as making a hotel booking, or a more complex task involving planning and reasoning, such as planning a family holiday, negotiating a contract, or managing a disaster scenario.”

Apart from that, there exist more subtypes of dialogue systems that are used to point out specific features. Some authors use the term *conversational dialogue system* to emphasise a very human-like behaviour and unrestricted dialogue. Jokinen and McTear (2009) use speech as a special characteristic and define a *spoken dialogue system* as a system that “*enables a human user to access information and services that are available on a computer or over the Internet using spoken language as the medium of interaction*”. In the context of this thesis, we consider three types of task-oriented dialogue systems.

Control system: e.g. for controlling the lights in a room by speech.

Question-answering system: the user has a question (open or closed domain) that will be answered by the system.

Information-seeking system: the system asks questions in order to provide task-related information.

However, we set a special focus on *information-seeking dialogues* or *dialogue-based information systems*.

Convention 8 (*Dialogue-based Information System*)

A dialogue-based information-system is a machine that offers task-related information and corresponding actions by verbally communicating with a human user.

This means we delimit from conversational/entertainment, task planning and tutorial dialogue systems. In this type of dialogue system, the system asks questions in order to gain information that is necessary to fulfil the user’s request. The best known example is a booking system. Here, the system has to collect information like date, number of persons and destination before it is able to present corresponding trip proposals that may be booked afterwards.

3.2.1 Architecture and Components

A (spoken) dialogue system consists of five components, as you can see in Figure 3.1 that pictures the flow of information through the system. The *automatic speech recogniser* (ASR) transforms spoken language into text, that then has to be interpreted by a *natural*

language understanding (NLU) component. The *dialogue manager* (DM) decides how to react on any user request and is able to interact with an *information source* (back-end). Depending on the request and the result from the back-end request, the system can generate a question or an answer. The specific formulation is created by a *language generator* (LG) and is eventually *synthesised* (TTS, text-to-speech).

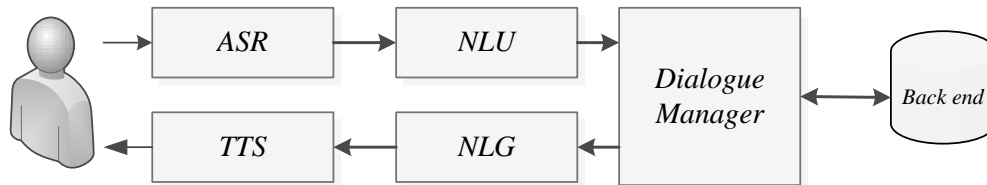


Figure 3.1: Components of a dialogue system

Depending on the focus and complexity of the system, components may vary. Some simple systems use predefined texts instead of a language generator or directly connect utterances with certain actions without performing a full semiotic analysis. If we are dealing with a text-based dialogue system, speech recognition and text synthesis are missing. In many cases, the dialogue manager is the heart of the system and connects every component. The DM decides – depending on the dialogue strategy, the current context and the user utterance – how to proceed. This may include the formulation of a back-end request (e.g. query a database or control external devices), the presentation of information, the creation of an error message or the generation of a question to get more information or to clarify the user’s intention in case of ambiguity.

3.2.2 Dialogue Strategy

After having introduced the basic components of every DS, we now focus on the dialogue manager and three different dialogue strategies.

- In a *system-initiative* dialogue, the system asks all the questions and the user is only able to answer. He cannot make own requests or influence the dialogue flow at all.
- In a *user-initiative* system the user makes all the requests and the system only answers. It cannot make own proposals or ask questions to the user.
- A *mixed-initiative* dialogue system combines the advantages of the former two types and allows the user to take over control by asking questions or by giving more information than has been asked for, while, at the same time, the system is able to make proposals and helps the user to reach his aim.

A system-initiative system is similar to a GUI-form. The system has total control of the dialogue by asking questions the user has to answer. There is no possibility for the user to influence the predefined dialogue flow. However, a major advantage is “that the

required vocabulary and grammar for each response can be specified in advance. In this way speech recognition and language understanding is constrained and are likely to be more accurate” (McTear, 2004). A typical dialogue would be:

Dialogue 3.1:

S: *Tell me your destination!*
U: *Aberdeen.*
S: *When do you want to start?*
U: *On the 8th of May.*
S: *When do you want to return?*
U: *On the 14th of May.*

The other extreme is user-initiative dialogue systems. Here, the user has the control, which leads to the fact that the system is not able to guide him to a certain aim. It's just a responder to the user's questions (e.g. a question-answering system), as you can see in the following example:

Dialogue 3.2:

U: *What is the capital of Scotland?*
S: *Edinburgh*
U: *How many people are living there?*
S: *420,893*
U: *Where can I see Dolly the sheep?*
S: *At the National Museum on Chambers Street*

In a mixed-initiative dialogue system *“either participant can take the initiative to ask questions, initiate topics [and] request clarifications”* (McTear, 2004), as shown in the next dialogue:

Dialogue 3.3:

S: *Where do you want to go?*
U: *I'd like to go to Oban next Friday.*
S: *And how long do you want to stay there?*
U: *How will the weather be then?*
S: *It'll be sunny.*
U: *Alright, then I'd like to stay until Sunday.*

In this example we can see that the user gives more information than is asked for and hereby actively influences the dialogue flow since the system does not need to ask for the date anymore. Afterwards, instead of answering the question for the duration of the stay immediately, the user first wants to know how the weather will be in order to be able to decide how long to stay. Whereas the latter utterance really is a change of initiative, the former one is only an over-informative answer. This also influences the dialogue, but does not change the initiative. McTear calls this *limited mixed initiative*.

3.2.3 Dialogue Control

A dialogue manager is the central component that decides how to continue with the dialogue, i.e. how to react on the user's utterance. Possible reactions are:

- ask the user for more information
- clarify or verify the user's input
- give information to the user (answer)
- inform the user about an error
- some physical reaction (e.g. switch the lights on)

This requires different data models and dialogue control strategies that we will introduce briefly.

Finite-state-based Approach

The simplest systems are realised as *finite state automata*. Every system question is represented as a node in the graph and the edges represent the user's answers, i.e. every dialogue step consists of a node and an edge, as you can see in Figure 3.2 (Jokinen and McTear, 2009).

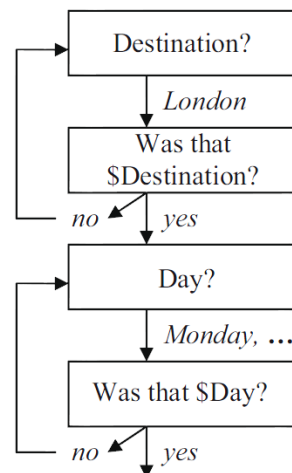


Figure 3.2: Finite state dialogue (Jokinen and McTear, 2009)

This only allows to create inflexible, static dialogues that follow a predefined order of questions. With the help of conditions it is possible to slightly adapt the dialogue towards the user. So, a confirmation question “*Was that \$Destination?*” could only be asked if the speech recogniser's confidence is below a certain threshold.

Frame-based Approach

A further developed approach of dialogue control is the *frame concept*. This concept relates to the well known printed forms with fields that need to be filled out. These forms have been adapted by the web and have later become the basis for many speech dialogue systems where the fields are presented to the user one by one (system initiative dialogue systems). In the context of speech applications, we speak of frames and slots (rather than forms and fields). Every slot represents the answer to a question, as is pictured in Figure 3.3.

$$\left[\begin{array}{ll} \textit{destination} : & \textit{London} \\ \textit{day} : & \textit{Monday} \\ \textit{persons} : & \textit{n/a} \\ \textit{start_date} : & \textit{n/a} \\ \textit{end_date} : & \textit{n/a} \end{array} \right]$$

Figure 3.3: Frame

After the user answers a question, the dialogue manager fills the corresponding slot, checks which questions are still unanswered (i.e. which slots are empty) and selects one of those to come next. The most important advantage is the possibility to be able to fill more than one slot with only one utterance. Furthermore, forms are order-independent and allow the user to decide when to give what information. This is also the requirement for open-ended questions like “*How may I help you?*”.

Plan-based Approach

Another approach is the *plan-based* dialogue control, which bases “*on the notion that agents in a dialogue are attempting to achieve a goal, which may be a physical state, such as arriving at some destination, or a mental state, such as knowing some information*” (Jokinen and McTear, 2009). This goal can be represented as a plan, i.e. a sequence of actions (which may depend on each other, i.e. the effect of subgoal 1 may be a precondition for subgoal 2) that lead from the current state to the goal. In the following example we can see a dialogue between a customer and a travel agent. In this plan-based system, the agent’s goal is to help the customer to book a flight. The agent might reason that it needs information about the flight to be able to book it. Knowing the month is not sufficient, so the simplest way to find out the exact date is to ask the client (Jurafsky and Martin, 2000). Consequently, the subgoal **ask customer** might be added to the plan.

Dialogue 3.4: (Jurafsky and Martin, 2000)

C_1 : *I need to travel in May.*

A_1 : *And, what day in May did you want to travel?*

C_2 : *OK uh I need to be there for a meeting that’s from the 12th to the 15th.*

A typical example for a plan-based approach is the *Belief-Desire-Intention Model* (BDI), where each agent decides what to do next based on his “*current beliefs about the domain, including nested beliefs about shared knowledge, and the discourse obligations*” (Jokinen and McTear, 2009). “*Beliefs are propositions taken to be true by an agent; desires are [...] goals that the agent wishes to achieve [and] intentions are actions the agent intends to perform*” (Larsson, 2002). This can be represented by a set of action schemas, that consist of preconditions, effects and a body. In the following example (Jurafsky and Martin, 2000) an agent A books a flight F for client C: The agent has a set of *true beliefs*

BOOK-FLIGHT	(A,C,F)
Constraints:	$Agent(A) \wedge Flight(F) \wedge Client(C)$
Precondition:	$Know(A, depart - date(F)) \wedge Know(A, depart - time(F)) \wedge$ $Know(A, origin(F)) \wedge Know(A, flight - type(F)) \wedge$ $Know(A, destination(F)) \wedge Has - Seats(F) \wedge$ $Want(C, BOOK(A, C, F)) \wedge \dots$
Effect:	$Flight - Booked(A, C, F)$
Body:	$Make - Reservation(A, F, C)$

Figure 3.4: Action scheme (Jurafsky and Martin, 2000)

(he *knows* facts) and the client has the desire to book the flight. By making a reservation the flight is booked and C’s desire is accomplished.

Information State Approach

A very flexible approach that is not limited to only one strategy is the *information state approach*, which “*allows specific theories of dialogue to be formalized, implemented, tested, compared, and iteratively reformulated*” (Traum and Larsson, 2003). Traum and Larsson define the information state of a dialogue as “*the information necessary to distinguish it from other dialogues, representing the cumulative additions from previous actions in the dialogue, and motivating future action*”. The information state theory consists of several components (Traum and Larsson, 2003):

- *Informational components*: participants, user models, beliefs, intentions, etc.
- *Formal representations* of the informational components as lists, typed feature structures, etc.
- *Dialogue moves* that trigger the update of the information state (including NLU and NLG)
- *Update rules* that govern the updating of the information state and select a particular dialogue move to perform given conditions on the current information state

- *Update strategy* for deciding which rules to apply

First of all, the developer has to decide *what* information he wants to model (e.g. according to the BDI model). Next, he has to define *how* to model them, i.e. which data types to use. The resulting data structure is referred to as the *information state* and, as the central part of all information, can be thought of as a complex frame (of course with far more information than in the frame-based approach). The specification of dialogue moves introduces an abstraction layer for the many different natural language utterances that correspond to the same update. For every type of update that the system should perform there needs to be at least one move. Often, speech-acts are used to tell apart different types of moves (e.g. *ask* or *answer*). Afterwards, by defining update rules, it can be formalised how to update the information state. Every rule consists of conditions and effects. Also the selection of the next move is part of this step. Finally, a strategy for how to apply these rules is needed, e.g. only apply the first rule or apply each rule.

3.2.4 Characteristics of Natural Dialogue Systems

“Natural dialogue systems are better dialogue systems”. This claim leads to many discussions. In this section we explain what a natural dialogue system is, clear up the question whether we need it and describe some features that we believe are necessary to create a natural dialogue system. Stent (2001) states, when speaking of *natural dialogue*, *“we mean that we are interested in how to automatically plan and produce dialogue behaviour that, at least in particular types of situations, seems to the human user of a dialogue system as though it could come from another human”*. Sometimes the possibility to interpret colloquial language, ellipses, injections and abortions/repairs is also required in natural dialogue systems. In practice, natural dialogue systems are often able to process more complex constructions than most current dialogue systems. The realisation of human-like dialogues requires features like:

- the use of natural language
- mixed-initiative: every partner can influence the progress of the dialogue
- flexibility: every partner can express information in a way as he wishes (more sophisticated NLU component)
- cooperativity and feedback: ambiguities and misunderstandings are solved, every partner indicates whether he understood what the opponent said

In this thesis, when speaking of a natural dialogue system, we refer to the following convention:

Convention 9 (*Natural Dialogue System*)

A Natural Dialogue System is a form of dialogue system that tries to improve usability and user satisfaction by imitating human behaviour.

The following two dialogues convey exactly the same information. The first dialogue only recognises single words that exactly match the question. The formulation is very machine-like and only one piece of information can be processed at a time. The second dialogue makes use of an open-ended question, uses more colloquial formulations and is able to understand complex information, e.g. Easter is converted into a date and the number of persons does not need to be asked because the user already gave this piece of information in his first utterance.

Dialogue 3.5:

S: *Tell me your destination!*
U: *Paris*
S: *Did you say Paris?*
U: *Yes*
S: *Departure date?*
U: *April, 20th*
S: *Return date?*
U: *April, 24th*
S: *How many persons?*
U: *Two*
S: *I will now list three hotels.*

Dialogue 3.6:

S: *How can I help you?*
U: *I'd like to go to Paris with my wife.*
S: *Oh, Paris, what a nice city. And do you already have a date in mind?*
U: *Easter would be great.*
S: *Ok, um, let me see.*
S: *I have fou... well... I have found three wonderful hotels.*

Human-likeness is connected with a more complex formulation compared to a simple command-and-control system. But the recognition and interpretation of complex sentences is of course much more error-prone than the selection of words from a list. Thus, critics of natural dialogue systems (NDS) argue that dialogue systems with sophisticated features lead to more ambiguities, because they evoke high expectations and encourage the user to use complex language that the system may not be able to understand. A system that formulates colloquial and very human-like sentences like “*Well, so where do you wanna go?*” and only understands single city names, will have trouble understanding the answer “*I'd like to go to Spain with my wife*”. This will result in frustration and in people claiming that they prefer standard systems (“*Tell me the name of the city where you want to start.*”) since they are more reliable. The utterances may not sound as human-like as the first request, but they influence the user to only say words the system is able to understand (e.g. *Madrid* instead of *Spain*). Hence, it is very important to always keep production and interpretation capabilities at the same level.

Some developers claim that reliability is more important than naturalness. Usability is characterised by success, time and simplicity and not by the level of anthropomorphism. For this reason, current dialogue systems often still use DTMF-based input or single word recognition. They offer almost 100% recognition rate, though being very unnatural. The user reaches his aim, but the way to get there is very tedious. He has no possibility to communicate with the system as he likes but instead has to follow rigid rules and machine-like instructions. Although this seems very limiting, Stent (2001) claims that humans adapt to their communication partner, i.e. they also adapt to the system and its

style of communication. The dialogue system is just seen as a tool (or in the interface metaphor), where we just have to learn how to use it. However, we think that the use of a system that behaves like a human does not need to be learned, because humans know how to communicate and expect a speech-based system to behave according to their experience of verbal communication. Although current systems have difficulties with complex human behaviour, we believe that this direction of research is highly beneficial and will lead to more effective and better accepted dialogue systems. Now we describe some features and techniques that are necessary for realising a NDS.

Mixed Initiative

As already introduced in Section 3.2.2, the *initiative* refers to the active partner of the talk, i.e. the dialogue partner who asks questions or gives commands and makes the other partner react. In contrast to human-human-communication, in man-machine-communication there exist two extrema: user-initiative systems are completely passive and only react. They don't ask questions themselves (e.g. Question Answering Systems). The other extreme is system-initiative or directed dialogue systems. They ask questions until they have enough information to fulfil the user's request. The user has no possibility to influence the dialogue flow (e.g. IVR systems). This often leads to disappointment because the user gets bored and feels slowed down, because it is not possible to skip dialogue steps or to give more than one piece of information in a single utterance. The combination of both extrema leads to mixed-initiative systems, which we regard as a minimal requirement for every NDS. They give the user more flexibility and enable him to influence the dialogue flow while the system still tries to guide and help him by actively asking questions. A common way of giving the user as much flexibility as possible is *open-ended questions* like "How may I help you?". After the analysis of the user's answer the system only asks for missing information which leads to an efficient talk.

However, the decision if a dialogue system is mixed-initiative is difficult because – especially in industry – developers consider it not as the pure change of initiative. Often, every system that implements a frame-based approach is called mixed-initiative, because the user can decide what information to give when and does not need to follow a pre-defined dialogue flow. This type of system is called *limited mixed initiative* (McTear, 2004).

Adaptation

We differentiate between two types of adaptation: adaptability and adaptivity. Adaptable systems can be configured by the user before or on runtime. He could change the synthetic voice, alter the dialogue strategy (directed vs. mixed-initiative), change the language or modify the preferred level of politeness. An adaptive system, on the other hand, continuously analyses the dialogue and adapts automatically to the situation. In case of many misunderstandings the system could change from mixed-initiative to a directed dialogue style which increases the probability that the system understands the user because every single information is asked for explicitly. Another possibility of adap-

tation is the automatic change of formulation. There exist people who prefer to speak with the system in a telegram-like style whereas others prefer long sentences.

This form of adaptation is an important feature for natural dialogues because also humans communicate by this means, i.e. depending on the dialogue partner they change the style of formulation. This enables the system to adapt to the user and behave as he expects, which in consequence leads to a more satisfied user.

Implicit Confirmation

Since there exists no visual back-channel and speech input is, at the same time, more error-prone than the input in graphical user interfaces, we need to find ways of ensuring the correctness of information. We can use two different confirmation strategies. If we use explicit confirmation, the user input will be repeated and explicitly asked for its correctness (“*Where do you want to go?*”, “*London*”, “*Do you want to go to London?*”, “*Yes*”). In the case of implicit confirmation, the last user answer will be integrated into the next system question (“*Where do you want to go?*”, “*London*”, “*When do you want to go to London?*”). This type of feedback enables the user to intervene in case of a false recognition without increasing the number of dialogue steps.

Verification Questions and Resolution of Ambiguities

Language is inaccurate and ambiguous which easily leads to misinterpretations and obscurities. But also background noise and clipped speech complicate recognition. In order to not continue with false information, it is important to verify the response if the recognition confidence falls below a certain threshold (“*Did you say Boston or Houston?*”) or if the utterance is ambiguous (“*Light on!*”, “*Do you want to switch on the ceiling lamp or the desk lamp?*”). Since in the first case the recognition will not improve if we ask this question, it is a good idea to artificially introduce a difference between the two similar words (“*Did you say Boston, Massachusetts or Houston, Texas?*”).

Correction

The possibility to correct own statements or assumptions of the system is essential for a natural dialogue. Apart from a direct correction as in Dialogue 3.7 there are also indirect corrections that look like an answer to the question but at the same time correct a piece of information (Dialogue 3.8). This type of utterance is particularly difficult to process because it consists of two functions (new information and correction of existing information).

Dialogue 3.7:

- S: *When do you want to go to Munich?*
- U: *But I want to go to Cologne!*
- S: *When do you want to go to Cologne?*
- U: *Tomorrow.*
- S: *And with how many persons?*

Dialogue 3.8:

S: *When do you want to go to Munich?*

U: *I want to go to Cologne tomorrow.*

S: *Alright, tomorrow to Cologne. And with how many persons?*

But also partial corrections are not easy to interpret as the following example shows.

Dialogue 3.9:

S: *So you want to travel with two persons on 5th March 2013 to Paris, right?*

U: *Yes, but with three persons.*

In this case the verification question is misleadingly answered with “yes” although not everything is correct. The user’s intention is the correction of the number of persons while the rest is confirmed. Apart from corrections that are caused by misunderstandings, also shifts of opinion may be the reason for a correction.

Dialogue 3.10:

S: *When do you want to travel?*

U: *From 1st until 10th August.*

S: *And with how many persons?*

U: *No wait, one week later would be better.*

Simple systems would regard this as uncooperativity and repeat the last question without considering the correction. In the worst case, the word “one” would be interpreted as an answer to the question for the number of persons.

Over-Informativeness

Over-Informativeness describes the situation when the user gives more information than the system has asked for. Mostly, the user anticipates the next system questions and hereby reduces the number of following dialogue steps, as the next example shows. The left dialogue is the standard way of dialogue design, whereas the right one minimises the number of turns.

Dialogue 3.11:

S: *When do you want to depart?*

U: *Tomorrow.*

S: *And what’s your destination?*

U: *Cologne.*

Dialogue 3.12:

S: *When do you want to depart?*

U: *Tomorrow to Cologne.*

In the second case, the system has to maximise the answer domain, because the answer can not only contain matching information but also facts that relate to a question that has not been asked yet (e.g. city names). Many current dialogue systems just ignore the extra information and still ask the question “*And what’s your destination?*” although the user just told the system where he wants to go, which is of course not very human-like.

Negations

Another feature of human-like communication is negations and restrictions. If a user is asked for his destination, he could answer “*To Spain, but not to Mallorca*” or reply to a question for special wishes with “*I don’t like smoking rooms*”. Simple keyword-spotting systems would incorrectly search for trips to Mallorca or select only smoking rooms, because they don’t have a deep linguistic knowledge. Often, the first mentioned term will be replaced by the last mentioned term of the same type (e.g. Spain is replaced by Mallorca). This results in systems that offer answers that the user has explicitly excluded.

Discourse and Anaphora

Anaphora are commonly used to connect statements throughout subsequent sentences, especially in narratives. Starting from simple references like “*Phil buys a car. It’s red.*” up to more complex statements like “*Sam gave Susan the pad. She opened it, noted some words and gave it back to him.*”. Some references¹ are only interpretable with the help of world knowledge:

“*We gave the bananas to the monkeys because they were hungry.*”
“*We gave the bananas to the monkeys because they were ripe.*”

Only monkeys can get hungry, whereas only bananas can be ripe. But there are also examples that may not be interpretable at all or only with the appropriate context:

“*We gave the bananas to the monkeys because they were here.*”

As a zookeeper it may be obvious that the pronoun *they* refers to the bananas because the monkeys are always there, but from a syntactic point of view there is no way to tell if it refers to the monkeys or the bananas. But also in dialogues we use anaphora. After switching on the light, the user may say “*Switch it off again*” or after answering the question how many children will come along he might ask if he could get a cot for *them*.

Interpretation of Colloquial Language

One of the biggest problems and at the same time a very important feature is the understanding of colloquial language and formulations that require context knowledge. Here, we do not only refer to slang but also to complex utterances. When asking for the number of persons, the answer “*two*” is much more easier to interpret than “*I and my wife*”. Also statements like “*... and my son as well*” are hard to interpret but often occur in human-human-communication. In this connection, also colloquial exclusions like “*A smoking room would be bad!*” should be mentioned.

¹examples taken from <http://www.linguist.univ-paris-diderot.fr/~amsili/Ens14/pdf/ho-anasem-1.pdf>

Formulation and Language Generation

When speaking about natural formulation, we refer to a style that resembles the way of human sentence production. Utterances should be formulated in an understandable, unambiguous, informative way and should be adapted to the situation. They should follow the Gricean Maxims and only use words that can be understood by both the user and the system. Because most people judge a system by the user interface, its careful design is very important. Speech interfaces are at even greater risk. Apart from the functionality, the style of communication has a big influence on the user. Some users may prefer short and telegram-like sentences while others like long sentences. Some users prefer to be addressed formally while others like an informal style. For offering every user his personal user interface, we need adaptive systems. Language generation methods allow us to generate user tailored utterances rather than using predefined ones. This also helps to prevent monotony because the system is able to generate slightly different formulations every time the user calls (random prompting).

Social Behaviour

Closely connected to the formulation of system utterances is the design of the system's social behaviour. The ability to greet the user, to say good bye, to reply to a thank, to express opinions (“*Good choice! This is a very nice destination.*”) or to engage in small talk does not add to the functionality of the system, but lets the system appear more human-like and thus increases its acceptance.

Speech Recognition and Synthesis

For a natural dialogue system, the speech recognition engine should be able to correctly recognise utterances with delays, repetitions, direct corrections, filling particles (um, err) and people clearing their throat. The recognition of an utterance like “*I'd like to go to Ber, *harrumph* , to Berlin tomorrow at, um, 10 no at 9*” is a challenge but common in human-human-communication. Also, in contrast to written language, speech does often not conform to grammar rules which additionally complicates recognition and interpretation. Another feature that every NDS should support is *barge-in*. It allows the user to interrupt the system when he wants to say something before the system prompt is finished. On the synthesis side there are also possibilities to increase human-likeness. Instead of using robot-like synthetic voices, modern voices sound quite natural and also support the generation of laughing, coughing and other sounds.

3.3 Classification of Functions in Dialogues

One of the most important goals of every dialogue system is the analysis of what the user intends to do. Before dealing with the actual content, the functional class of the utterance is an essential step in the analysis process and crucial for the dialogue system to interpret the user utterance correctly which allows the system to react appropriately.

3.3.1 Dialogue Acts

Dialogue Acts are a well-known method for expressing the function of an utterance. They base on speech acts which have been introduced by John L. Austin and have been extended by John R. Searle, as you have already read in Section 2.4.1. Recall that by speaking we commit acts. When you say “*Nice to see you*” you actually greet, saying “*Mind the speed limit*” is a warning and by saying “*I name this ship Queen of California*” you change things of the real world by just saying words. Austin distinguishes three levels: the locutionary act refers to the performance of the utterance, the illocutionary act is the intended meaning, i.e. what we *do* by saying words (e.g. warn, promise, greet), and the perlocutionary act is the effect of the utterance (e.g. scaring, delighting).

When we speak of speech acts, we mostly refer to the illocutionary layer. Searle distinguishes *primary* and *secondary* illocutionary speech acts. The secondary illocution refers to the literal meaning of the utterance, while the primary illocution refers to the actual goal. The utterance “*Could you please hurry up a bit?*” seems like a question (secondary illocution), but actually is a request (primary illocution). In the context of a speech processing system, the most important thing is to understand the goal of the user. The *natural language understanding unit* should generalise utterances into an abstract representation. Hence the primary illocution is a vital feature for expressing the user’s goal.

In a dialogue we have to consider more than a single utterance. This becomes important when we interrogate the user because his answer is only comprehensible in relation to the question. Consequently, speech acts have been extended to *dialogue acts*, which have been introduced by Harry Bunt in 1994. They deal with the issue “*how the recognition of some sort of act can aid in the interpretation of a user’s utterance*” (Webb, 2010). However, in contrast to speech acts, there is no global consent on the specification of dialogue acts. Stolcke et al. (2000) describe a dialogue act as “*approximately the equivalent of the speech act of Searle (1969), the conversational move of Power (1979), or the adjacency pair part of Schegloff (1968)*”. They further describe it as “*a tag set that classifies utterances according to a combination of pragmatic, semantic, and syntactic criteria*”.

The most prominent work, which we are going to introduce now, has been done by James Allen and Mark Core resulting in *DAMSL - Dialog Act Markup in Several Layers* (Core and Allen, 1997; Allen and Core, 1997) and Harry Bunt who developed *DIT - the Dynamic Interpretation Theory*. These models have later been combined to DIT++.

DAMSL

Dialogue acts have been used in many projects like *Verbmobil*, *Amities* and *Sundial*. *Verbmobil* uses more than 40 dialogue acts, 18 of them being on the top-level (e.g. thank, greet, bye, accept, reject, introduce or confirm). As all projects make use of similar, but still different, dialogue acts, the Discourse Resource Initiative created a domain-independent dialogue act tagset called DAMSL (Dialogue Act Mark-up in Several Layers). One of the best known projects that implemented DAMSL is TRAINS. DAMSL “*defines a set of primitive communicative actions that can be used to analyze dialogs*”

(Core and Allen, 1997). Core and Allen extended the speech act theory by removing the limitation of only being able to assign a single label to each utterance. It allows “*multiple labels in multiple layers to be applied to an utterance. Thus an utterance might simultaneously perform actions such as responding to a question, confirming understanding, promising to perform an action, and informing*”. The DAMSL tagset is characterised by the differentiation between backward-looking and forward-looking acts: the forward communicative functions are similar in style as the actions of traditional speech act theory whereas the backward communicative functions show how the current utterance relates to the previous dialogue (Core and Allen, 1997).

By this means it is possible to assign several acts to a single utterance. This is necessary because the reaction “*Do you still have free tickets for tomorrow evening?*” to the question “*How can I help you?*” is both an answer (backward-looking) and a question (forward looking). Moreover, DAMSL consists of a third part called *utterance features* that captures “*features of the content and form of utterances*”, i.e. the information level dimension (task, task management, communication management), the communicative status (abandoned, uninterpretable) and syntactic features (conventional, exclamatory).

DIT++

Bunt points out that the term *dimension* is not correctly used in DAMSL. While in DAMSL a dimension is just “*a label for a cluster of mutually exclusive tags*” (Bunt et al., 2010) (statement, info-request, agreement, understanding, answer, ...), Bunt refers to dimensions as “*various types of semantic content*”. He explains (Bunt, 2006) that every entity needs to have exactly one value for every dimension. DAMSL does not hold this demand. Moreover it allows the definition of conceptually wrong combinations (assert and question the truth at the same time) although a “*multidimensional annotation scheme should ideally guide annotators in not considering impossible tag combinations*” (Bunt, 2006). He also concludes that questions and answers can not only be about the task but also about the communication itself, i.e. there are functions that “*do not belong to any particular dimension*”.

Thus, DIT++ (Bunt et al., 2010), an advanced combination of DIT and DAMSL, has been developed. DIT++ is described as “*a comprehensive, application-independent system for the analysis of human and human-machine dialogue and for annotating dialogue with information about the communicative acts that are performed by dialogue segments*” (Bunt, 2010). Most dialogue act schemas annotate utterances by the intended effects (questions, invitations, promises) or their form (repetitions, hesitations, openings). The problem of classifying by form is that utterances with the same form can have different intentions or even one utterance can have different possible intentions depending on the context (“*Why don't you play with us?*”, “*Why don't you go and buy a beer?*”, “*Why don't you start?*”). So Bunt follows a strictly semantic approach. He distinguishes between *domain specific functions* that “*can only be used in a particular dimension*” and *general purpose functions* that “*can be applied to any kind of semantic content*” (Bunt, 2010). Examples for the former are *stalling* and *pausing* which are functions of the *time management* dimension. The *acceptance* or *release* of a turn clearly are *turn management*

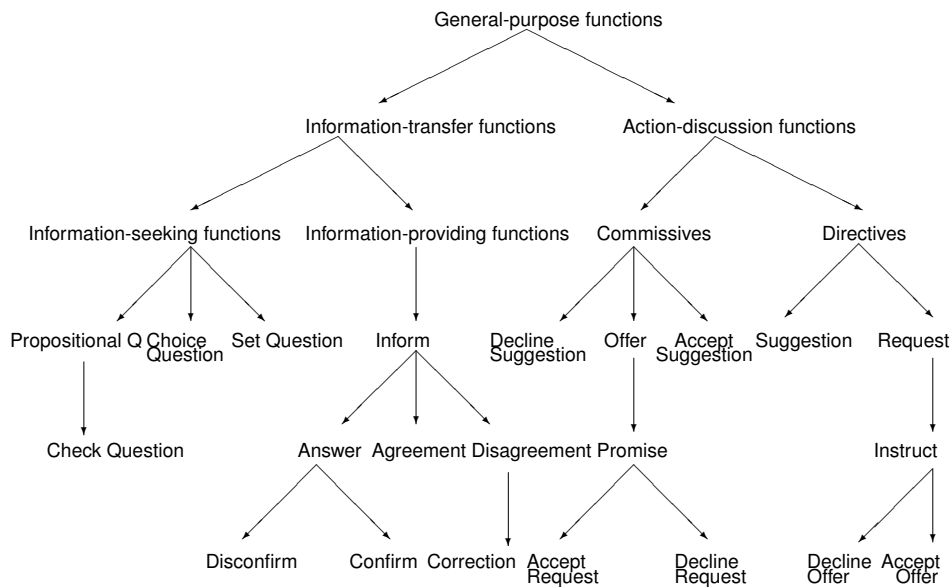


Figure 3.5: Taxonomy of general purpose functions in DIT++ Bunt et al. (2010)

functions. Other dimensions are task, feedback (correct understanding), discourse structuring, partner communication management and social obligation management. General-purpose functions, which you can see in Figure 3.5, are applicable in any dimension. The following example shows possible variants of the inform-act in different dimensions (Bunt et al., 2010).

- “*We will be open this Sunday.*” [Task]
- “*I didn’t hear what you said.*” [Auto-Feedback]
- “*You misunderstood me.*” [Allo-Feedback.]
- “*I have nothing more to add.*” [Discourse Structuring]
- “*I need a moment to check this.*” [Time Management]
- “*I think Peter should continue.*” [Turn management]
- “*I’m very grateful for you help.*” [Social Obligations Man.]

As you can see, the general-purpose functions can be divided into:

- information-seeking functions
- information-providing functions

- commissive functions
- directive functions

Information-seeking functions refer to utterances that put pressure on the addressee to provide information that the speaker wants to know and that that he believes the addressee knows (e.g. propositional question, set question). *Information-providing* functions are the counterpart and provide information that the speaker believes to be true and the addressee does not know (e.g. inform, correction). *Commissive* functions commit the speaker to a certain action (e.g. offer, promise) and *directive* functions order the addressee to carry out a certain action (instruct, request).

3.3.2 Interaction Patterns

In his work about rapid prototyping of SDS, Denecke (2002a) focusses on the separation of generic dialogue processing algorithms from task-/language specific knowledge sources. The system's architecture consists of three layers, one of it being the *interaction pattern layer* that provides task- and language independent patterns that can be instantiated by the *dialogue control layer*. Interaction patterns “are sequences of utterances designed to obtain information to be added to or to be removed from the discourse” (Denecke, 2002a). There are four different types of interaction patterns:

Question: ask the user for information that will be added to the discourse, e.g. “*Would you like A, B or C?*”.

Undo: is triggered by user utterances like “*undo*” or “*No, not A*” and removes information from the discourse.

Correction: removes and adds information to the discourse, e.g. “*I said A not B*”.

State: does not change discourse information but influences the dialogue flow by utterances like “*repeat*” or “*help*”.

Every pattern can be instantiated in various shapes, i.e. it can lead to different dialogues because the dialogue develops over time and depends on the user's answers. The following two dialogues (Denecke, 2002a) are instantiations of the same interaction pattern:

Dialogue 3.13:

S: *Would you like A, B or C?*
U: *A*

Dialogue 3.14:

S: *Would you like A, B or C?*
U: *A*
S: *Please say again!*
U: *A*
S: *Did you say A? Please say yes or no.*
U: *Yes*

As you can see, the concrete realisation differs and can also include verification questions. In contrast to dialogue acts, interaction patterns are not used to tag utterances but rather define the interaction abilities of the system. Here, interaction patterns make use of speech acts, as the following question pattern shows (Denecke, 2002b):

$$\langle I, SA_I, SA_O, F, F' \rangle, \text{ where } SA_I = \{act_question\}, SA_O = \{act_answer\}, \\ F = \text{semantic content of the expected answer}, F' = \emptyset \quad (3.1)$$

This question interaction pattern is described by the initiator I (user, system), a set of input speech acts SA_I that the initiator is allowed to instantiate the pattern with, a set of output speech acts SA_O , a set of feature structures F that should be added to the discourse and a set of feature structures F' that should be removed from the discourse. The instantiation with a disjunctive question could lead to the realisation “*Would you like to go to a Chinese restaurant in Squirrel Hill, Shadyside or East Liberty?*” (Denecke, 2002b). Thus, interaction patterns describe if information should be added to or removed from the discourse and the instantiation describes how to do it.

3.3.3 Adjacency Pairs

“*People can’t say just anything during a conversational exchange; their utterances are constrained in various ways by the context, most notably the utterances of other interactants*” (Holtgraves, 2001). This constraint can be described by adjacency pairs. Adjacency pairs are two subsequent turns of different speakers where the second one relates to the first one and is also constrained by it. A greeting is usually answered by a greeting, a question by an answer and an offer by an accept or decline. It would be unusual to respond to “*Would you like a cup of coffee?*” by saying “*Hi*”. So, Schegloff (2007) states that adjacency pairs are typologised and that every part of the pair must be from the same type. Consequently, “*adjacency pairs reflect obvious regularities in peoples’ talk; people return greetings, answer questions, accept offers, and so on*” (Holtgraves, 2001). However, in human communication subdialogues are very common:

Dialogue 3.15:

- A1: *Where do you want to go?*
- B1: *How’s the weather in Paris?*
- A2: *Sunny.*
- B2: *Ok, Paris.*

As you can see, the first two turns do not form an adjacency pair. Instead, A1 and B2 and B1 and A2 relate to each other. Further critique concerns dialogues that do not form regular adjacency pairs at all, e.g. a greeting that is not returned or a question that is not answered. However, apart from some exceptions, adjacency pairs build a basic description for regular communication and can serve as rules of how to respond to different types of utterances.

4

Related Work

In this chapter approaches that are similar to the aims of this thesis are being presented. The related work is divided into four fields of topics referring to the main aspects of this thesis: First, studies about the communication between man and speech interfaces are introduced in order to investigate what features a user-friendly and natural speech dialogue system should possess. Afterwards, work on the modelling of dialogue steps and their corresponding questions and answers is analysed. Furthermore, work on the variation of the linguistic style is considered which aims at systems that adapt to the user to embody a more natural behaviour. Finally, development environments for creating dialogue systems are presented.

4.1 Studies towards the Interaction of Humans with Speech Interfaces

The user interface of an information system is crucial for its pleasant and efficient operation. Especially in the context of dialogue systems, it influences the impression that the user has of the system and its abilities. Because there are hardly any studies on the user interaction with spoken dialogue systems, also related studies, that clarify how users prefer to interact with speech interfaces of various types and that identify the effects an adaptation of the linguistic style may have, are considered.

Effects of message style on users' attributions toward agents, Brennan and Oheari

Brennan and Ohaeri (1994) look at the question whether spoken dialogue systems with an anthropomorphic utterance style appear more intelligent than those which make use of a telegraphic and command-like style. For this purpose, they conduct a Wizard-of-Oz study where 33 users make reservations with air carriers. Every participant needs to complete six tests where he is randomly assigned to a group in which the system uses an anthropomorphic, fluent or telegraphic formulation.

The authors did not find any hints on the hypothesis that a more human-like formulation lets systems appear more intelligent. However, they warn that extremely human-like

systems may also be counter-productive as they tempt the user to make indirect requests that are harder to process.

Generation of Output Style Variation in the SAMMIE Dialogue System, Kruijff-Korbayova et al.

Kruijff Korbayová et al. (2008) explore the connection between the assessment of a spoken dialogue system and the formulation of the system's utterances with respect to formality. They aim at learning whether an adaptation of the interpersonal style does influence the opinion about the system. This study is part of the TALK project which resulted in the SAMMIE dialogue system. SAMMIE is a multimodal interface for an MP3-player in cars. It can vary the following characteristics: personal / impersonal style, telegraphic / non-telegraphic style, reduced / non-reduced referring expressions, choice of near-synonyms and use of adverbs.

In the experiment, two versions of the dialogue system have been compared under simulated driving conditions. 28 participants have been asked to search songs by their title or certain characteristics and manipulate the playlist accordingly, both by using the speech interface. Every task has been formulated once in a personal style (active voice) and once in an impersonal style (passive voice).

Kruijff Korbayová et al. (2008) did not find any evidence that a change of style in the personal layer (active / passive voice) influences the user's opinion about the dialogue system.

Human-Likeness in Utterance Generation: Effects of Variability, Hjalmarsson und Edlund

Hjalmarsson and Edlund (2008) analyse the effect of human-like system utterances. They conduct a study where they simulate a natural spoken dialogue system by a human. Eight ten-minute dialogues between humans (secretary and customer) have been recorded and transcribed. Two out of them have been selected and the voice of the secretary has been replaced by a synthesised one. This is the basis for two different dialogue examples. While the utterances of the first dialogue have not been altered at all, the utterances of the second one have been artificially restricted in their expressiveness without changing the meaning. This should make the dialogue appear more machine-like. 23 candidates had to evaluate the audio files of these dialogues. They believed to hear recordings from a real SDS and had to evaluate them according to human-likeness, politeness, efficiency, intelligence and understanding. Finally, the participants had to state which dialogue they prefer. They attributed the unchanged dialogue significantly more human-likeness, more politeness and a higher intelligence. But also in terms of efficiency and preference, the original dialogue led to better results. The artificially restricted dialogue only had a small advantage in the understanding category.

Finally, the participants have been asked to rank the evaluated features by their importance. Although human-likeness and politeness got the smallest scores, they still average at 3 out of 5 points which underlines the importance in view of the fact that an efficient

dialogue system that understands the user can only be realised with these features. A friendly spoken dialogue system with a bad speech recognition would be useless.

The authors conclude that men don't have problems to accept human-like systems. Moreover, systems with utterances that make use of the full spectrum of human conversational behaviour appear more intelligent and polite.

What is a Robot Companion – Friend, Assistant or Butler, Dautenhahn et al.

Dautenhahn et al. (2005) conduct a study about the perception and attitude of users towards robot companions. The study focusses on the question whether humans accept robots in the household, what tasks they could undertake, and which form of social behaviour would be best. Because the interaction with the robot is performed by speech, this part of the study is of special concern. 28 participants have been asked in what style the robot should communicate. 71% state, that the robot should communicate in a human-like style. It has to be noted, that the system at hand also provides a visual channel to give feedback to the user. Moreover, the robot may appear more human-like due to its embodiment.

Summary

None of the studies directly covers the analysis of the users' wishes regarding the features of spoken dialogue systems. Instead, it is examined whether the change of certain parameters of a dialogue system has influence on the user's assessment. Brennan and Ohaeri (1994) find that human-like formulations don't make a dialogue system appear more intelligent. Also Kruijff Korbayová et al. (2008) don't find evidence that a change of the grammatical voice (active / passive) influences the user's opinion. Hjalmarsson and Edlund (2008), on the contrary, come to the conclusion that human-like systems do not have a negative influence and instead appear significantly more intelligent and polite. In a survey, participants state that politeness and human-likeness are important to them. In their study, Dautenhahn et al. (2005) refer to the communication with robot companions and find that 71% prefer a human-like style.

None of these studies directly questions what features a good dialogue system should possess. Instead, existing and simulated systems are used to analyse the influence of specific parameters. Two out of the four studies result in the observation that human-like systems are not more accepted, but also not regarded as being worse than today's systems. The authors of the other two studies come to the conclusion that users actually prefer human-like systems. The differences in the results can be explained by the fact that the authors interpret the term *human-likeness* differently. Whereas Hjalmarsson and Edlund (2008) use genuine human utterances in their study, Kruijff Korbayová et al. (2008) only focus on the adaptation of active and passive voice to make an utterance human-like.

Since we do not know any study that addresses the original question and the studies at hand do not lead to a clear result, it's reasonable to conduct an own user study.

4.2 Modelling of Dialogues

Dialogue modelling is a topic with different facets. In linguistics and psychology, researchers model the behaviour of humans who engage in a dialogue. In computational linguistics most attention is drawn on the dialogue management, i.e. how to decide which question comes next and how to react on errors. Other researchers use dialogue acts to describe the functions or aims of utterances in a dialogue. However, in this section we concentrate on the elements needed in order to describe a dialogue and its single dialogue steps (questions and answers). Only with a specification of the dialogue elements we are able to create flexible and adaptive natural dialogues as this is a prerequisite to correctly use language understanding algorithms, language generation methods, speech recognition software and access back-ends.

ROBODIMA: A Dialog-object-based Natural Language Speech Dialog System, Quast et al.

Quast et al. (2003) describe a speech dialogue system toolkit that enables a developer to create natural dialogues by defining lexica with an attached *semantic type hierarchy* and specifying how to get semantic type information from the user. The model consists of dialogue objects, actions and a lexicon. A dialogue object is a representation of a real-world object like a hotel or an mp3-player. Dialogue objects are linked to each other and build a semantic type hierarchy. Every such object is part of a domain and can be characterised by several features which are called *slots* (e.g. city or street). In order for the dialogue manager to be able to ask the user for information, prompts and queries need to be specified in configuration files. After all necessary information has been gathered, the dialogue manager will execute the actions which are associated with every dialogue object. The combination of dialogue object and action is referred to as *frame*. Apart from frames, the developer needs to specify lexica, which are defined as lists of words with every list belonging to one or more semantic types. A probabilistic framework is used to find out which dialogue object and action a user utterance belongs to. The user has the possibility to fill any slot at any time and is therefore able to actively influence the dialogue flow. This enables him to engage in a very natural dialogue: the user may ask for the time while the system wants to get information about his destination. Moreover, the dialogue manager is able to ask disambiguation questions or to relax constraints when the query is too restrictive. The authors claim that by defining dialogue objects, actions and lexica, most tasks can be modelled. However, they point out that it is not possible to encode deeper domain-specific knowledge.

TrindiKit: Task oriented instructional dialogue toolkit, Larsson et al.

TRINDIKIT (Larsson et al., 2000) is a dialogue theory independent toolkit for building and experimenting with dialogue move engines and information states. Although TRINDIKIT focusses on dialogue management and the implementation of different dialogue strategies, it also describes a method for specifying questions, answers and the corresponding actions

for different dialogue steps.

The information state describes the information that is stored by a dialogue system. It is an abstract data structure that can be accessed by different modules (e.g. speech recognition, input interpretation, language generation) and that interfaces different resources (e.g. lexica, databases). Together with interface and resource variables, the information state is called Total Information State. The Dialogue Move Engine is responsible for selecting the next dialogue step (utterances are connected with one or several moves) and updating the information state. This is done via update rules that consist of conditions and operations on the information state. A control algorithm connects all modules and decides which one needs to be called in which order (e.g. select, generate, output, update, input, interpret, update).

Dialogue moves (e.g. ask, assert, request, confirm, greet) are abstract descriptions of utterances and combine a speech act type and information about the content (e.g. `ask(λ x.destination(x))`). These moves are collected in a dialogue plan and model the system's questions, actions and responses.

Queen's Communicator, O'Neill et al.

The QUEEN'S COMMUNICATOR is a Java-based, object-oriented dialogue manager that has been incorporated into the DARPA Communicator architecture. It separates domain-specific from generic dialogue behaviour. While generic confirmation strategies "*determine the system's behaviour across transactional domains*", domain-specific experts "*guide the user towards completion of particular transactions*" (O'Neill et al., 2003). The generic behaviour is controlled by the dialogue manager, which generates system utterances that mostly consist of an implicit confirmation and the next system question. The dialogue manager relies on specialised subclasses – the domain experts – that handle the domain-specific behaviour. The experts contain user-focussed rules that trigger responses to "*specific confirmed combinations of information supplied by the user*" and database-focussed rules that represent recovery strategies (e.g. constraint relaxation) that are applied if a database request fails.

The most important contribution of the Queen's Communicator is the *confirmation status* that indicates the confidence of the user-supplied information. Information that is not corrected after an implicit confirmation is assumed to be confirmed. Only if all slots are confirmed, the system will complete the transaction. Moreover, an object-oriented approach facilitates the separation of domain-specific and generic behaviour.

Jaspis, Turunen et al.

JASPIS is a Java-based "*open framework for adaptive speech applications*" that "*adapts to a user and an environment*" (Turunen and Hakulinen, 2000). A special focus is set on the development of multilingual applications. JASPIS consists of several modules that manage communication, information, presentation and the dialogue itself. The presentation manager provides dynamic, multilingual speech outputs based on conceptual system messages. The information manager coordinates the information access that is

based on a shared blackboard architecture. *Agents* are used to create more adaptive systems by providing the actual functionality of a module and are selected by *evaluators* depending on the current situation (e.g. the dialogue strategy can be changed by the evaluator by selecting an agent that implements a mixed-initiative instead of a system-driven strategy).

Dipper, Bos et al.

DIPPER “*is a collection of software agents for prototyping spoken dialogue systems*” (Bos et al., 2003) that “*comprises agents for speech input and output, dialogue management, and further supporting agents*”. This allows a plug-and-play integration of third-party modules that all use different programming languages across a distributed environment. Moreover, it includes an own dialogue manager that is based on the information-state approach and aims at taking away the complexity of TrindiKit. The author claims that the construction of a simple dialogue system requires little effort. It could consist of the “*Nuance speech recognition agent, the DME, and a synthesizer*”. Afterwards, the information state and the update rules need to be defined. The major differences to TrindiKit are the variable-free update language, the removed distinction between update and selection rules, and the open agent architecture that allows interfacing other components of the dialogue system.

Storyboard Design, Thalheim et al.

A *web information system* (WIS) is a database-backed information system that is realised and distributed over the web (Schewe and Thalheim, 2005). *Storyboards* are a methodology that was created for the design of large-scale, data-intensive web information systems. The design of such systems requires anticipation of the user’s behaviour. This problem is addressed by storyboarding. It describes the ways users may choose to interact with the system. The terminology refers to the area of films and plays. A storyboard consists of three parts (Ma et al., 2008): The stories, which are navigation paths through the system, the actors which comprise users with the same profile, and tasks which link activities (goals) of the actors with the story space, which is a container for the description of the stories. Subgraphs of the story space are called scenarios. Every action can be equipped with pre- and postconditions or triggering events. This allows us to specify under which conditions an action can be executed. The core of the story space can be expressed by a directed multi-graph, in which the vertices represent scenes and the edges represent actions by the users (e.g. navigation). Scenes can be encapsulated to build a hierarchy. They include a task, actors, a context and a specification.

Storyboarding focusses on the business and the conceptual layer of the five level abstraction layer model. The strategic layer is used to describe the system’s intention in a general way (mission statement). The business layer deals with user profiling and the design of the application story. It specifies the information from the strategic layer by describing stories that symbolise paths through the system. These paths depend on the users’ behaviours. In the conceptual layer, *scenes* in the storyboard are analysed and

integrated. The design of abstract media types supports the scenes by providing a unit that combines content and functionality. The presentation layer associates presentation options with the media types. In the implementation layer, physical aspects like setting up database schemata, page layout and the realisation of functionality by script languages are addressed.

Storyboard design focusses on the navigational aspect of information systems. However, user modelling and the definition of information units (dialogue steps or scenes) are a concern, too. Although Storyboards refer to the design of WIS, it may be interesting to see if the approach is useful for modelling speech dialogues.

Summary

There is not much work about the structure of dialogues and the combined modelling of input, processing and output.

Quast et al.'s *DIALOGUE OBJECTS* come closest to this aim. They describe hierarchic entities that are linked to each other. Every object consists of slots and is associated with an action that the system should execute. This model is very similar to ordinary frames with the extension that they are described by a semantic type hierarchy and link to corresponding actions. However, prompts and queries are not defined within the dialogue objects themselves but need to be specified in configuration files. Larsson et al.'s *TRINDIKIT* focusses on implementing different dialogue strategies based on information states. Single dialogue steps are defined by dialogue moves, which combine information about the speech act type and the content. There is no integrated description that specifies the input and output of every step. Instead, *TRINDIKIT* focusses on a detached architecture: independent modules link to the dialogue moves and are responsible for their correct processing. Also Thalheim's *STORYBOARD DESIGN* does not perfectly match the requirements. It refers to the modelling of web information systems and makes use of screenplay metaphors. However, it deals with GUI-based dialogue steps (scenes) and does not focus on speech applications. Also the *QUEEN'S COMMUNICATOR* aims at creating dialogue systems with special attention on confirmations. However, it is not clear how the dialogue is defined exactly. There are no examples given on how to specify all the information for the dialogue and the corresponding actions. This is also true for *JASPIS*. Moreover, it is not mentioned whether language generation methods are used, which would be very beneficial for the creation of adaptive and multilingual dialogue systems. Finally, *DIPPER* focusses on the interaction of different components and is an improvement of *TrindiKit*.

We can see that most work deals with the modelling of dialogue engines instead of the description of a dialogue definition that can be automatically processed by the engine.

4.3 Linguistic Style Adaptation

The way how a system behaves and articulates is crucial for the impression that the user gets of the system. The linguistic style that the system makes use of defines its character

and essentially influences the user's opinion about the dialogue system. Most current spoken dialogue systems only pursue a rigid and monotonous style that does neither adapt to the user (social distance, age, language) nor to the domain (banking application / cinema ticket booking). Language generation techniques can help because the dialogue designer does not need to write every utterance manually but instead defines generation rules that create utterances based on abstract descriptions according to a given style and context. The following sections give a short overview of approaches to the adaptation of the linguistic style.

Politeness and Alignment in Dialogues with a Virtual Guide, de Jong et al.

Language alignment happens, according to Jong et al. (2008), “*automatically in dialogues between human speakers*” and does not only refer to the syntactic level but also to the linguistic style. The authors explain that the ability to adapt increases the believability of a virtual agent, makes him appear more socially intelligent and gives him a personality. Consequently, Jong et al. describe a model that adapts to the user with respect to politeness and formality.

They develop a virtual guide that gives directions in a theatre by speech and gestures, can answer questions about the location of objects and displays routes on a map. The model contains three dimensions: politeness, formality and T-V distinction. While politeness refers to the choice of sentence structures, formality is influenced by the choice of words. Furthermore, we have to distinguish between a formal and informal addressing¹ of people. This characteristic has influence on both formality and politeness. Nevertheless, in de Jong's model, varying formality and politeness values have no effect on the T-V distinction in order to prevent a constant switching between both extrema.

This model leads to polite answers if the question is formulated in a polite way and leads to impolite answers in the other case. The variation of formality is based on a *shaded lexicon* that has a formality score assigned to each phrase. This database is used for the analysis of the formality of the user utterance and also for the generation of system answers. The politeness adaptation is based on a study by Roel Vismans, who identified politeness scores for different (Dutch) formulations of the same command (“*close the door*”), and on politeness indicators like sentence structure and modal particles.

POLLY: Politeness for Language Learning, Gupta et al.

Gupta et al. (2007) describe a system called POLLY (Politeness for Language Learning) that “*combines a natural language generator with an AI Planner*” in order to realise Brown et al.'s politeness theory. It engages with the user in a collaborative, task-oriented dialogue that aims at conveying fun to learn English as a foreign language. The authors believe that it is hard to learn politeness as a non-native speaker and that virtual environments can help to overcome this problem.

A *content planner* determines the correct plan for the fulfilment of the goal and chooses a verb and the direct object accordingly. The *sentence planner* subsequently creates a

¹German: Sie/du, French: vous/tu

deep syntactic structure depending on the desired speech act, that is then transformed into a sentence by the *surface realizer*. The speech acts can be realised in several variations in order to represent the desired level of politeness by adding lexical items like *please*, *if you don't mind* or *mate* to the syntactic structure. These politeness formulas can be divided into four categories: *address form* (“*mate*”), *abstracting the subject* (“*someone should*”), *softeners* (“*if you know*”) and *additives* (“*for me*”).

The dialogues that have been generated by POLLY were finally evaluated by candidates from two different cultural backgrounds (British and Indians) in terms of the perception of politeness. The participants of the web-based survey had to rate the politeness of the utterances on a five points Likert scale. In order to represent social distance, some of the dialogues were described as a conversation between friends and others as a conversation between strangers. The authors indicate that utterances from a friend were rated more polite than utterances from strangers. Also softeners increase the level of politeness whereas the abstraction of the subject results in smaller politeness values. The results show that Brown and Levinson's strategies have a significant effect on the human perception of politeness. It is interesting to see, that results differ with regard to cultural background: “*Indians rated the sentences as overall much more polite than British*”. Moreover, the indirect strategy has been perceived as being most impolite, which is contrary to Brown and Levinson's theory.

CRAG: Individuality and Alignment in Generated Dialogues, Isard et al.

Isard et al. (2006) state that “*it would be useful to enable dialogue agents to project, through linguistic means, their individuality or personality*”. They refer to Nass and Lee (2000), who found that user responses are influenced “*by whether the agent's linguistic personality matches [...] the personality of the user*”. Isard et al. present CRAG, a system for creating dialogues between two simulated agents. It is able to generate linguistically distinguishable dialogues about movies and is at the same time able to align the formulation to its interlocutor.

CRAG makes use of the OpenCCG framework which expects a logical form as input and creates a list of possible sentences as output. The authors “*massively over-generate paraphrases*” and choose the candidate – with the help of n-gram language models – that matches the agent's personality best. These models were trained with data from a study from Nowson (2006) about the analysis of blogs. The data has been divided according to five personality dimensions (extraversion, neuroticism, openness, agreeableness and conscientiousness) into *high*, *medium* and *low*. A cache language model that uses the last generated utterance as reference simulates the alignment.

In order to simulate the dialogue, two characters are parametrised by means of the personality dimensions. Apart from the definition of the agents, also aspects of the topic (e.g. movies) and the respective opinions about them are stored in the agenda, e.g. (*soundtrack, positive*) or (*main actor, negative*). The agent with the higher level of extraversion initiates the dialogue. Depending on the settings, CRAG is able to generate fictional dialogues with different linguistic styles and different values for the level of alignment. There is no interaction with the user. Instead, the dialogue only relates to the interaction between the two agents.

PAULINE: Planning And Uttering Language In Natural Environments, Eduard Hovy

Many current dialogue systems “*produce the same output to all hearers in all circumstances*”. Hovy’s work (Hovy, 1987, 1990) from the late 80s can be seen as one of the first steps towards the generation of individual, user-dependent texts and to answer the question how pragmatic and stylistic aspects influence the generation of texts (Hoepfner, 1990). Apart from the well-known problems of *what to say* and *how to say something*, he adds the question “*Why should something be said?*” and thus focusses on a pragmatic view.

Basis for his work is the description of three events: The destruction of a shantytown in Yale, a hypothetical presidential nomination, and a fight between two people. Any of these events is reported on from different points of view. The influences on these reports are then analysed individually by author and point of view. Based on these findings, Hovy developed PAULINE – a system that can generate paragraphs that differ in terms of point of view, content and style under consideration of pragmatic categories like personal characteristics of the interlocutors (e.g. emotional state, relationship of the interlocutors), goals of the speaker with respect to the hearer, and the conversational atmosphere (e.g. available time, type of the room). This leads to 23 different parameters each of which may have three different values. Depending on the chosen plan (*describe*, *relate* or *convince*) the text is complemented with phrases like “*not only X but Y*” or adverbs like “*really*” or “*only*”.

Flexible Natural Language Generation in Multiple Contexts, Cullen et al.

Cullen et al. (2009) present a template-based approach (MINTGEN) to natural language generation that allows the generation of phrases across pre-defined business domains and *registers* (the tone of language). They “*created an ordered hierarchy of mini-templates and sparse domain-specific terms that forms the basis of an intuitive object-based toolkit*”. For asking how long the user wants to stay at a hotel in a normal English register, the dialogue manager needs to send the following command to the language generator: [info_request][gain_info][duration][-][hotel][-][normal][english] which is then processed stepwise by several modules until the utterance “*How many nights do you want to stay at the hotel?*” has been created. A change of the register to *formal* would result in the utterance “*For how many nights does one wish to be accommodated at the hotel?*”.

Summary

Jong et al. describe a user-oriented politeness and formality model that is applied in a virtual guide. However, the model mainly relates to system answers and not to the generation of questions in different styles. Gupta et al. implement Brown and Levinson’s politeness theory applied in a language learning application. The authors primarily cope with different formulations of requests depending on the social distance between the interlocutors. Isard et al. present a system that generates a complete dialogue about

movies with the possibility to influence the level of alignment. Although the alignment in terms of choice of words is clearly recognisable, there exist difficulties in projecting the personality. Moreover, the generated dialogues give an artificial impression. Hovy's work shows that it is possible to generate texts in a way that they express a certain point of view as well as the relationship to the hearer. Furthermore, external influences like time pressure or an unpleasant atmosphere can be simulated. However, Hovy refers to the expression of point of views in continuous text (e.g. journal articles) and not to the variation of politeness in dialogues.

Consequently, Jong et al.'s work comes closest to the respective subgoal of this thesis. However, the question how we can vary the linguistic style of questions remains unanswered.

4.4 Modelling of Development Environments for Dialogue Systems

Despite numerous approaches to the design of natural dialogues, current dialogue systems and development environments do not implement them. By analysing the state of the art, we found that most systems only use state-based and static dialogues with key word recognition. The reason for that is most likely basic and restrictive development environments that only allow the creation of simple dialogue systems. This also explains the discrepancy in terms of quality between research prototypes (created manually) and current commercial dialogue systems (created with IDEs). Because the development of sophisticated dialogue systems is expensive and time-consuming, development environments that facilitate their creation and support the designer in developing natural dialogues are very important. In this section we present work about the development of spoken dialogue systems.

DUDE: A Dialogue and Understanding Development Environment, O. Lemon et al.

Lemon et al. describe a graphical development environment called DUDE (Lemon et al., 2008; Lemon and Liu, 2006) "*which allows non-expert developers to produce complete spoken dialogue systems*". This saves costs and time and is an alternative to Voice XML applications that are "*difficult to build and maintain, because developers must anticipate and represent every possible dialogue path*". Lemon et al. use the information state update approach and describe the *advanced dialogue tools* (ADT) which allow the creation of dialogue systems based on business user resources. These tools can access a database and a process model that describes the structure of the application. Moreover, they offer language models and grammars and the developer may choose between different dialogue strategies.

Although the user does not directly get in touch with Voice XML, the ADT automatically generates it based on the database and the business process model (BPM). BPM is usually used for specifying the structure of tasks (e.g. the order of questions that the human agent asks in a call centre). However, BPM can also be processed by a

machine. This requires the extension of the model by possibilities to interpret the user utterance and to define the dialogue strategy as these tasks are naturally performed by a human operator and thus are not specified separately. Domain-specific interactions are specified with the help of BPM, while the dialogue manager contains general, domain-independent aspects of the dialogue (e.g. clarification questions). Moreover, the ADT compile all database contents into grammars. The activation of tasks is realised by spotter phrases that are also transformed into grammars so that the recognition of the words *hotel*, *room* or *sleep* activates the respective hotel-booking subtask from the BPM. If a spotter phrase is associated with several tasks, a clarification dialogue is initiated. This allows the usage of the “*How may I help you?*” – approach instead of asking multiple-choice questions. The definition of the system questions is done statically. Furthermore, the developer defines a template phrase for implicit confirmation and for a clarification question. Answer types support the automatic generation of grammars by reading all city names from the database if the answer type *CityName* has been specified.

ARIADNE: Rapid Prototyping for Spoken Dialogue Systems, M. Denecke

Denecke (2000) describes an “*integrated development environment for spoken dialogue systems*” which is based on reusable dialogue packages that can be defined with a graphical user interface. Each of these packages consists of an assembly of data sources (task model, domain model, dialogue goal, grammars). The task model is similar to a form-based approach that is filled stepwise during the dialogue together with constraints “*stating the minimal amount of information necessary to execute the task*”. Also, the form contains information about the current dialogue state as it describes the discourse. The dialogue strategy is specified by a Prolog-like declarative programming language.

The dialogue developer has to follow four steps with the help of the graphical user interface, in order to specify the various knowledge bases. First, the domain model is described by a simple class hierarchy, which is also the basis for the grammar generation. The domain model describes the attributes of the objects and their possible values. A pizza can be delivered with different kinds of toppings whereas pasta can be served with different sauces. Both are available in various sizes and both have a price. This makes it possible to create a class hierarchy where *pizza* and *pasta* are derived from *food*. In the next step, the developer defines the dialogue goal by specifying a typed feature structure that consists of attributes from the domain model. Afterwards, the grammar needs to be defined. A generic, abstract food grammar can parse sentences of the form “**X with Y**”, e.g. “*Pasta with tomato sauce*” or “*Pizza with pepperoni*” and is specialised depending on the type. Semantic annotations prevent the combination of incompatible objects (e.g. pizza and sauce). In the last step, the database access is specified.

SpeechBuilder: Facilitating Spoken Dialogue System Development, E. Weinstein

Weinstein (2000) focusses on the simplification of creating mixed-initiative spoken dialogue systems. The author developed the rapid prototyping software SPEECHBUILDER, which allows the dialogue developer to specify linguistic information about domains and to build speech interfaces to them. Like most of the related work, also SPEECHBUILDER focusses on the generation of grammars. However, the developer only needs to define semantic concepts and to provide exemplary user utterances as well as corresponding system actions. The definition of the action is realised by CGI parameters. The sentence “*turn on the lights in the kitchen*” is represented by `action=set&frame=(object=lights,room=kitchen,value=on)`. This representation is created by the interpretation component and is based on a parse tree transformation. If no complete parse tree is available, a concept spotting approach is used. The *concept keys* define classes of semantically equivalent words (room = kitchen | living room | dining room). *Actions*, on the other hand, define classes of functionally equivalent sentences (set = turn the radio on in the kitchen please | can you please turn off the dining room lights). The system response that follows the action is defined statically and does not make use of language generation methods.

DialogDesigner: A Tool for Rapid System Design and Evaluation, Dybkjær et al.

Also Dybkjær and Dybkjær (2005) conceptualise a system for the development and evaluation of spoken dialogue systems. With the help of the DIALOGDESIGNER, dialogue structure and system prompts can be defined. For this purpose, a new group that consists of several conceptionally related dialogue states needs to be created. Every state is associated with one or more statically defined system prompts. These states are connected by transitions and thus lead to a state-based dialogue. The speciality of the DIALOGDESIGNER is the integrated Wizard of Oz environment, which allows to simulate a man-machine-dialogue. In this case, the system does not use its understanding capabilities but instead lets the wizard choose the next state based on the user utterance.

CSLU: A Platform for Research and Development of Spoken-Language Systems, R. Cole et al.

The CSLU (Center for Spoken Language Understanding, Oregon) platform is one of the first development environments for dialogue systems. Because it has influenced many works in this area, we want to briefly present this platform. The CSLU RAPID APPLICATION DEVELOPER (McTear, 1999a,b; Cole, 1999) has primarily been created for teaching and allows the creation of state-based dialogues over a graphical user interface. Single dialogue steps are represented by objects that realise special tasks like speech recognition, speech synthesis and the execution of actions. Every dialogue starts with a start object and is completed by a goodbye object. Apart from a grammar, the developer can also provide the system with a list of terms that should be recognised. Moreover, special grammars (e.g. for the recognition of numbers) are provided. The definition of

system prompts is realised by static text which can be synthesised or played back by pre-recorded audio files. The possibility to specify subdialogues enables students to create simple and well structured dialogues, e.g. to order meals or to retrieve the balance from your account. Furthermore, it is possible to automatically integrate repair dialogues as soon as the recognition confidence falls below a certain threshold.

Summary

When comparing the systems, it can be noted that all graphical development environments model state-based dialogues. This is done by running through four steps:

- Definition of task or goal
- Definition of the domain (properties of the objects and their relations)
- Definition of the grammar
- Definition of the system prompts

In DUDE and SPEECHBUILDER, grammars are defined indirectly by answer types and semantic concepts which saves the developer a lot of time. ARIADNE partly derives the grammar from the domain model. None of the presented systems enables the developer to adapt the dialogue to the user. Neither language generation nor a change of the dialogue style is considered. This results in a dialogue that is identical for any user and may seem unnatural due to language related conflicts between system and user (e.g. formal vs informal language). Moreover, none of the systems supports over-informative answers.

All of the presented tools focus on a graphical modelling of the dialogue flow with a simple possibility to integrate speech recogniser, speech synthesis and databases in order to facilitate work processes by dictating a methodology that supports the developer in creating natural dialogue systems. This is also in line with the goals of this thesis. However, we do not focus on the dialogue flow but instead aim at formalising system questions and their answers.

4.5 Conclusion

With respect to the goals of this thesis, we identified four areas of related work. First of all, we reviewed studies about speech-based human-computer-interaction. None of the existing studies answers the questions what type of SDS users prefer, which features they like or dislike most and how important politeness is in speech dialogues. However, there is interesting work about the effects of human-like systems towards the user. On the one hand, Brennan and Ohaeri (1994) and Kruijff Korbayová et al. (2008) found that human-like formulations do not increase the intelligence that is attributed to the system. On the other hand, Hjalmarrsson and Edlund (2008) disagree and claim that human-like speech interfaces are observed as more intelligent and more pleasant. We can summarise that polite and human-like systems do not have a negative influence on the dialogue and

that according to Dautenhahn et al. (2005) most people prefer a human-like style and regard politeness as very important. In order to get to know how people want to speak with a machine, we have to conduct our own survey.

For being able to build systems that address the identified user preferences, we analyse related work concerning the modelling of dialogues. Quast et al.'s (2003) dialogue objects are an interesting approach of extending the frame concept with semantic types and actions. However, they do not integrate the specification of prompts and queries. Although Larsson et al.'s TrindiKit and Thalheim's Storyboard Design focus on different aspects, the description of dialogue moves and scenes gives valuable hints for developing a new model to define dialogue steps.

An important area for the realisation of natural dialogues is linguistic style adaptation. Jong et al.'s politeness and formality model is similar to one of the goals of this thesis. However, they refer to system responses instead of system questions. Further work models social distance by implementing Brown and Levinson's theory of politeness (Gupta et al., 2007), presents a system that aligns the formulation of two agents (Isard et al., 2006), and describes a method to generate texts from different points of view (Hovy, 1987, 1990). There couldn't be found any work on the modelling of politeness and formality of questions in dialogues.

Another goal of this thesis is to integrate the results into an environment that facilitates the development of natural dialogue systems. Current approaches make use of answer types (Lemon et al., 2008; Weinstein, 2000) or a domain model (Denecke, 2000) to facilitate grammar development. But none of the reviewed systems adapts the linguistic style to the user, integrates language generation facilities or provides sophisticated NLU modules.

Part II

About the User's Perspective on Natural Dialogue Systems

Synopsis

Our aim is the improvement of spoken dialogue systems. So we need to know what users think of current dialogue systems and what they wish for the future. As already mentioned in the Related Work, we have not found any existing studies that answer this question. For this reason, we will conduct our own studies.

In Chapter 5 we first describe a user study about the interaction with speech interfaces. We classify users by means of their style of interaction and their choice of formulation. In this study, which is conducted as a Wizard of Oz experiment, users have to solve tasks like looking for the price of a book by speech only.

Afterwards, we describe a survey on spoken dialogue systems and the users' expectations regarding style and usability in Chapter 6. We ask the users for their preferred style and formulation and let them assess some example dialogues.

The results of these studies will be the basis for our work in the context of modelling natural language dialogue systems, which will be described in the next part. Hence, we conclude by formulating the requirements for a SDS from the user's perspective in Chapter 7.

It is a capital mistake to theorize before one has data.

(Sir Arthur Conan Doyle, 1859 – 1930)

5

User Study about the Interaction with Speech Interfaces

Interacting with a machine as if it were a human being is a long-cherished research objective. In the last years, especially in the field of call centres, many speech-enabled systems came onto the market. To avoid misunderstandings and usability problems, the design of a functional and effective speech processing system requires a careful analysis of the user's behaviour. We want to identify how people want to talk to computers and whether they want to talk to them at all. Therefore, we describe the setup and the results of a qualitative user study, which is realised as a Wizard of Oz experiment (WOZ).

In a WOZ experiment, a human operator (the wizard) is hidden behind a computer interface in order to simulate a conversation with the user, who believes to be interacting with a fully automated prototype (Quarteroni, 2008). This reduces the development effort and allows the simulation of systems that have not yet been implemented.

In this study, which is conducted together with the University of Rostock, we don't simulate a classical information system but instead imitate a cooperative agent that enables the user to pursue basic activities like reading and writing e-mails¹. We examine whether there is a preferred type of interaction for a speech-based interface and whether there are differences between experienced and inexperienced users or between blind / visually impaired and sighted users. While the colleagues of the University of Rostock are interested in the latter question, we focus on the former one.

5.1 Hypotheses

We hope to be able to differentiate the user behaviour by two dimensions: *mindset* and *phrasing*. The former describes the mental way of how people think about a SDS and how they operate it while the latter describes how people formulate their utterances when instructing the system. We divide the dimension *mindset* into *menu-oriented* and *task-oriented*. Someone who regularly uses computers and the Internet will have a complex mental model of their usage. Therefore, when using only speech to operate the system, the user may pursue a *menu-oriented* way just like when using keyboard, mouse and

¹Although we primarily refer to information systems in this thesis, the results of this study are also relevant because they provide interesting facts about how users behave and how they formulate their concerns in general.

screen. We assume users with a broad experience in using graphical user interfaces (GUI) to remember menu structures and even to use the exact words of the menu labels when addressing the system. Such users act in a *menu-oriented* fashion. Users with less experience do not have such a clear idea of the structure of applications, so we assume they will focus on the task and use more complex instructions (as they know from human-human-communication). We call this a *task-oriented* mindset.

We divide the second dimension, *phrasing*, into *commands* and *complex language*². Commands are instructions formulated as short utterances whereas complex language is characterised by instructions formulated in full sentences and resembles the communication style between humans. We expect the users to behave like this:

- Beginners interact in a task-oriented way and use complex language.
- Advanced users interact in a menu-oriented and command-based fashion.
- Both user groups will change their strategy over time into a task-oriented and slightly more command-based communication style in order to be more effective.

We think people with less experience don't know the exact menu labels and will therefore use more complex instructions, i.e. a more human-like communication style with full sentences, filler words and politeness markers instead of plain commands. So we expect a correlation between a task-oriented mindset and the use of complex language. We also assume a correlation between a menu-oriented mindset and command-based phrasing. This assumption is based on our consideration that people with a broad experience will have memorised the GUI (they are primed) and will follow the menu structure and even use the exact menu labels to interact with the speech interface. Naturally, menu items are commands and therefore lead to a command based formulation. Besides, experienced users are aware of the limited capabilities of many speech interfaces and computers in general, so that they do not believe a speech interface could work in any other way except for the one they know. In other words, we expect beginners to follow the human metaphor whereas experts supposedly follow the interface metaphor.

Many users tend to interact differently with a machine than with another human being. According to Brennan, the reason is not a general rejection of human-like communication with machines, but is based on failures and bad success when interacting with SDS. In order to prevent errors, we expect some users to adapt to the system on the assumption that the system is inflexible and only has limited understanding capabilities (Brennan, 1996).

We also expect a transition of both user groups towards a task-oriented mindset and a slightly more command-based phrasing. Task-oriented instructions are more effective than menu-oriented ones because not every single step needs to be mentioned and the communication partner derives intermediate steps (Instead of saying "*Open the mail programme and search for John in the contacts and then open a new message with his e-mail address as recipient*" it's more effective to say "*Please send a new e-mail to John*"). A

²We regard the term *natural* language as inappropriate because a command is also natural language

user who interacts in complex language will probably, after repeated usage, omit phrases of politeness and filler words as this reduces the time needed while leading to the same results. So he could slowly move towards a command-based formulation like “*New e-mail to John!*”.

5.2 Setup and Methodology

The experiment takes place in two different rooms. As sketched in Figure 5.1, the test candidate is sitting in a separate room and is equipped with a microphone to talk to the computer and with speakers to hear the system’s responses. There is no screen and thus no GUI. Furthermore, the user has no input devices to point at or type in anything. For enabling the wizard, who sits in another room, to hear the candidate’s spoken instructions, the audio data is sent to the wizard’s computer. In order to create and maintain the impression of a computer talking to the candidate, we decided not to let the wizard talk himself. Instead, the wizard uses an application that allows the management and synthesis of utterances, so that the system’s responses are sent back to the candidate as synthesised speech. The audio data (the user’s instructions and system responses) will be recorded for later transcription and analysis.

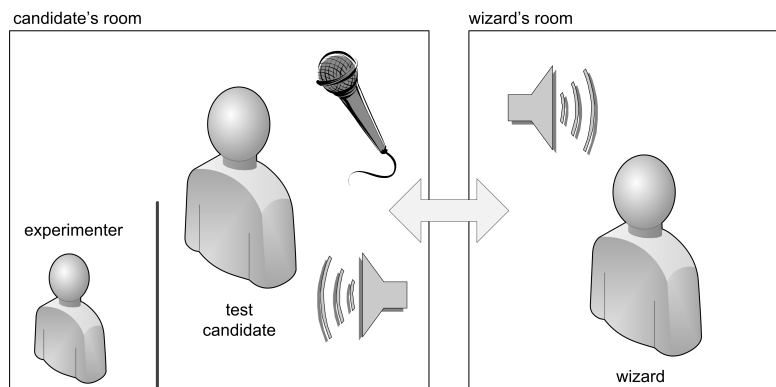


Figure 5.1: Wizard of Oz experiment

In order to collect both demographic information about the participants and their impression on the scenario, we prepared two questionnaires. One is filled in before the experiment and the other one after the experiment. The first questionnaire contains demographic questions about age and gender as well as questions about the participants’ visual faculty and their computer and internet experience. In the second questionnaire we ask the participants how they felt using the speech interface, whether they would use such a system in everyday life and in which situations they could imagine to do so. Furthermore, we ask whether the system was reacting in the expected manner and how they would wish the system to react in the future (only short keywords or human-like answers in full sentences).

For this qualitative study we acquired a total of 18 participants, 11 males and 7 females.

We classified three age groups: seven participants were between 20 and 40 years, another seven participants were between 41 and 60 years and four participants were older than 60 (three of them even older than 70) years. Most participants (14) had regular vision. Out of the four participants with low vision, one used a screen magnifier as an assistive device. Another participant used the computer through her husband, who reads e-mails to her and types in what she wished to reply. The rest did not use any assistive technology but two out of all participants (one with vision impairment and one elderly user) need to make fonts and icons on the screen appear as large as possible.

After the first questionnaire, the participants are introduced to the situation of sitting in front of a computer with an internet connection provided and all common applications installed. The only difference to a common workplace is that there is neither a screen, a keyboard nor a mouse but a microphone and speakers. Then, the subject is asked to solve two given tasks. In both tasks the subjects should check their e-mails. The received e-mail contains a task to perform an online search. The first task is to investigate the price of a book whereas in the second task the candidate shall look up the weather forecast. The search result should then be sent back to the sender. The subject is free in how to solve these tasks while the wizard tries to simulate the system the user expects to have. Both menu-oriented and task-oriented approaches are possible.

5.3 Results

We now present selected results of this study which are relevant to answer the question how users interact with SDS and how they want them to behave. We first categorise the subjects according to mindset and phrasing and then point out some observations before we generalise the results and conclude.

Interaction types

While evaluating the data and trying to classify the users with the help of the matrix described in the last section, we realised the need to extend the dimensions. The two categories for the mindset-dimension – menu-orientation and task-orientation – are not sufficient. Some subjects are not clearly menu-oriented but are obviously familiar with the used applications. They have a clear idea of the GUI but do not move along the exact menu structure. We call this an *application-oriented* behaviour.

The categories of the phrasing-dimension are not sufficient either. Some subjects clearly use commands, others use complex language, i.e. they utter full sentences including words of politeness and filler words. Yet others formulate their instructions somewhat in between these two classes. So we extend the phrasing dimension by *phrased commands*. Both the new categories and the classification of the subjects are shown in Figure 5.2. People who are using the exact words from the menu and who are acting in the corresponding steps, like starting the e-mail programme (“*start Thunderbird*”), opening a new mail (“*new mail*”) and focussing the subject text field (“*go to subject*”) just as they would do by using the mouse, interact in a *menu-oriented* fashion. They have a pro-

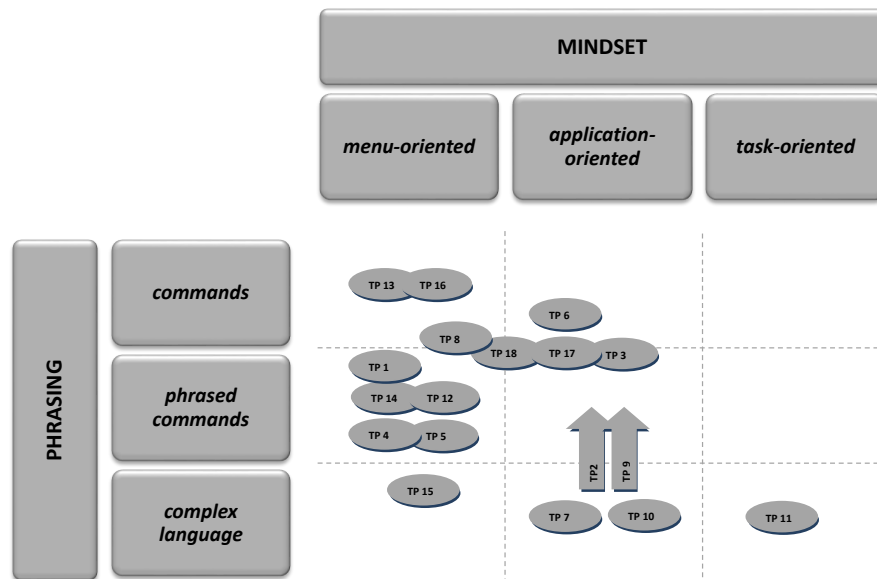


Figure 5.2: Results of the WOZ experiment

found knowledge of GUIs and computer operations. People who care about starting and quitting programmes but who do not follow the menu structure are called *application-oriented* users. People who are detached from the GUI and who are just focussed on the task to fulfil, interact in a *task-oriented* way. The reference to GUIs is strongest in the case of menu-oriented interaction and smallest in the case of task-oriented interaction. Along with the decreasing dependence from the GUI, the complexity by means of task-aggregation and thus reduction of the user’s workload increases. We consider a task-oriented mindset to be the most complex, most natural and most effective form of interacting with the computer by voice. Apart from the mindset, also the *phrasing* is an important description of the user’s behaviour. The simplest form are short *commands*, i.e. imperatives or nouns like “close”, “send and receive” or “Google”. The second form of expression is called *phrased commands*. They are also imperatives but expressed as short phrases like “open the browser”, “check mails” or “write new mail”. The most natural form are full sentences or colloquial expressions like “Please write a new mail to Gabi”. This class is called *complex language*.

Observations and Examples

In our study, the interaction types *task-oriented/commands* and *task-oriented/phrased commands* did not occur at all. The *task-oriented/complex language* type occurred once (TP11). This subject was totally detached from the GUI and fulfilled the tasks in one sentence: “Computer, Internet, check weather in Munich for the weekend and mail the

forecast to Gabi". Two of the test-candidates show, as expected, a transition from full sentences to phrased commands, as this is shorter and more effective. TP2 first uses full sentences like *"I want to look up the price of the book [...]"* but in the second task switches over to a list of commands like *"Put in password, check e-mail, new e-mail"*. We can observe a similar behaviour with TP9. In the first run, he uses phrases of politeness such as *"please"* and *"thanks"* several times, e.g. *"Please read first e-mail"*, *"Ok, thank you"* and *"Thanks. Well. Open Browser."*. He talks to the computer as if it were a human being (*"Do you have results?"*). But in the second run he omits phrases of politeness and uses short commands like *"Open e-mail programme"* or *"Open browser"*. As supposed, most users interact in a menu-oriented way (or at least in an application-oriented way if they are not too attached to the menu structure) and have a clear idea of the screen content.

Sometimes, a precise distinction between the categories *commands* and *phrased commands* is not easy. Subjects TP13 and TP16 clearly interact in a command-based fashion as they use one-word instructions like *"Firefox"*, *"E-Mail"*, *"Send"*, *"Books"* or *"Vendor"* whereas TP1, TP4, TP5, TP12 and TP14 use phrased commands like *"Send e-mail"*, *"open new e-mail"*, *"please send"*, *"please show price"*, *"open first entry"* or *"open a new tab"*. TP8 is placed in between these two categories as this subject changes categories during the tasks. He mostly uses phrased commands, but as soon as he gets to the point to fill in forms like in an e-mail, he uses one-word instructions like *"address"*, *"subject"* and *"text"*. In these situations the subject expects a response from the system indicating that it is ready, just like the cursor would blink as an indication that the user may type in some text. Also TP15 plays a special role since this subject is actually telling the system what she wants or what she does: *"I want to login [...]"*, *"Then I read [...]"*, *"Then I put in [...]"* or *"Then I close [...]"*. This candidate has more difficulties than other participants in instructing the computer and rather describes what she would do with a GUI.

Tendencies and Generalisation

The second questionnaire shows that most of the subjects can imagine to use a speech-operated system in their everyday lives. They would use it to check e-mails while driving a car, to look for a recipe while cooking or in general to search the Internet while their hands are not free. Surprisingly, most subjects believe that the system used for the study is a real system and are surprised how well it works. They are satisfied with the system's reactions and claim that it behaves like expected. However, opinions about the preferred interaction style vary. Some subjects, mostly the elderly, would like to experience a human-like reaction – they even would like to be asked *"How are you?"* – whereas other users, mostly experienced ones who use a computer very often, prefer short answers.

We now analyse the correlation between phrasing and mindset with an *association rule learner* algorithm and observe the following trends:

- menu-oriented users tend to use phrased or single commands
- all people who interact in a task-oriented way use complex language

- all people using phrased or single commands interact in a menu- or application-oriented fashion
- people using single commands or phrased commands never interact in a task-oriented way
- task-oriented people never use commands

These results prove that menu-orientation is strongly connected with a command-based formulation. The formulation of more complex utterances is associated with task-orientation. This is obvious because formulating complex tasks requires more words and is often connected with full sentences. This is why single commands and task-orientation exclude each other. Moreover, the formulation of full sentences in connection with a menu-oriented mindset is rarely used as it is ineffective.

In a second step we look for relations between the target attributes and the answers of the first survey (e.g. computer expertise). One of the hypotheses is that people, who have been using computers for several years, are more likely to have a menu-oriented mindset and to use a command-based phrasing than people with less computer experience. This hypothesis cannot be confirmed.

Most people have a particular idea of how a computer interface works in general. They have a well-practised method in mind (using a computer with mouse, keyboard and screen), which they transfer to the speech-operated system. When they are told they could have used a different type of interaction – for example a task-oriented approach – most subjects are surprised and claim they did not think of that possibility. This indicates that people are committed to their way to deal with a computer. On the other hand, some experienced subjects do not believe a speech interface to work with complex language because they already gained negative experience when using those systems.

We also observe that the more uneasy the person feels, the more natural the utterances are going to be, i.e. complex language increases with unease. But also complex tasks can lead to a change in interaction types. We notice this when subjects want to search for something and do not find a menu-structured way to solve the problem. Instead, they switch over to a task-oriented way, as can be seen in the following dialogue extract of TP 3.

Dialogue 5.16:

Usr: *To inbox!*

Wiz: *Programme opened.*

...

Usr: *Read mail!*

...

Usr: *Logout!*

Wiz: *Confirmed.*

Usr: *Internet!*

Wiz: *The browser is now open.*

Usr: *weltbild.de*

Wiz: *URL loaded.*

Usr: ***I am looking for the price of the book “The Wizard of Oz”.***

Wiz: *The book “The Wizard of Oz” is 13 Euros.*

Usr: *OK, thanks.*

Wiz: *You’re welcome.*

Usr: *Back to the inbox.*

5.4 Conclusion

In this study we have examined how users interact with computers using speech only. The results show that there are strong relations between phrasing and mindset. Most people – independent from age, gender, experience or profession – stick to the GUI. This can be explained by a mental conditioning: people are just used to this form of interaction. Surprisingly, even visually impaired people have shown this behaviour. Moreover, we have learned that most of the participants categorise natural, complex language as helpful. It was interesting to see that in case of unease or in the need to solve a complex task, users also switched to complex language. Furthermore, when explicitly offering the possibility to naturally interact with the computer, they were surprised and stated that this would be much easier than application- and command-oriented control. We suppose that many people are “blind” towards this form of interaction because they do not believe it to work.

You've got to start with the customer experience and work back toward the technology - not the other way around.

(Steve Jobs, 1955 – 2011)



Survey on Spoken Dialogue Systems

In this section we evaluate the results of a survey about the user expectations regarding the style of a dialogue system. We want to find out, which type of dialogue system users like most. Our assumption is that, in general, users prefer systems with human-like capabilities and a natural style. We also imagine that there are no clear relations between preferred style and age or background. We believe that every user is different and that we need systems that adapt to the users with regard to style and formulation.

6.1 Questions and Method

The survey is completely web-based and consists of 18 questions that can be seen in Figure 6.1. It is structured into four main categories:

- General questions (3)
- Experience with SDS (5)
- Opinions regarding style (6)
- Examples regarding style (4)

In the first category, we ask general questions regarding sex, age and computer skills. This is followed by questions about what users dislike about current spoken dialogue systems and how satisfied they are. The third category focuses on what users expect from future systems, if they are interested in polite man-machine-dialogues and if they prefer human-like sentences to short machine-like statements. In the last category we present some exemplary dialogues in different styles and let the users decide which one they like most.

6.2 Results

The survey has been completely answered by 194 participants. While the survey was active for about six months, 86% of all answers have been given in the first two weeks.

1. Gender
2. Age
3. Computer Skills (4 options)
4. Have you ever used a Speech Dialogue System (talking computers, e.g. hotlines)?
5. How satisfied have you been with the operation/usability? (0..5)
6. What did you dislike? (7 options)
7. What needs to be improved in today's systems or how are systems from television (e.g. Star Trek) better? (5 options)
8. Do you think that speech understanding systems are (in future) beneficial or do you think that these systems are superfluous? (4 options)
9. Independent of the current technology: Would you like to instruct the computer in short commands or would you rather use full sentences? (3 options)
10. Should a computer speak in full sentences or do you prefer short keywords as answers? (2 options)
11. What do you think of open questions? (questions without predefined answer like "How may I help you?") (2 options)
12. What do you think of directed dialogues? ("Say balance, change plan, or failure report") (3 options)
13. Should a computer be polite? (Should it greet you and say "please" and "thanks"? (3 options)
14. Are you polite yourself when talking to a computer? (Do you say please and thanks?) (3 options)
15. If you want to tell the computer to switch the light on, what do you say? (4 options)
16. If you tell a co-worker to switch the light on, what do you say? (4 options)
17. Which dialogue do you think is more pleasant? (2 options)
18. You want to know how the weather will be. Which dialogue system would you use? (3 options)

Figure 6.1: Survey questions

It was advertised among friends, colleagues, students and LinkedIn groups. 85.6% of the survey has been taken in German and the remaining 14.4% in English. The mean time for completing the survey was 5m52s, resulting in 16 seconds per question. In the following sections we will not give a complete report on every detail, but instead highlight interesting results.

6.2.1 Socio-Demographic

The gender distribution is almost equal with a slight majority of male participants (55.2%). The majority is aged between 19 and 35 (19-25: 25.3%, 26-35: 47.4%). There were only 0.5% under 19 and 3.1% older than 65. 14.9% were between 36 and 49 and 8.8% between 50 and 65. The distribution of male and female participants is very homogeneous throughout the age groups with a derivation of less than 5%. 41.2% of all participants are IT specialists or computer scientists, 50% use the computer at work on a daily basis and 8.8% use a computer only at home.

6.2.2 Experience and Satisfaction

90.7% of all participants claim to have used a SDS at least once in their life. On a scale from 0 to 5, the mean satisfaction was 2.63 with a standard deviation of 0.86. Asking for the main problems with dialogue systems resulted in the following top three answers:

- The *question-answer-game* was very tedious (62.8%)
- The system did not understand me (55.2%)
- The system offered too many choices (51.7%)

In relation to the last point, 45.3% claimed to have difficulties in identifying the correct menu item. Surprisingly, only 22.7% complained about systems that are only able to recognise numbers, while at the same time 38.4% complained about systems that only recognise single words instead of full sentences.

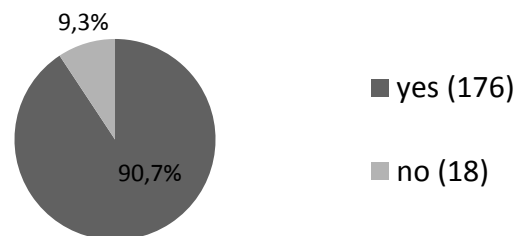


Figure 6.2: Have you ever used a SDS?

After asking for the problems users have with current dialogue systems, we asked what should be improved and where fictional systems from TV are better. 60.8% voted for a better speech recognition. This result has been expected, because 55.2% stated, that the system often does not understand the user. It has to be mentioned, that the participants might not have been aware of the difference between speech recognition and natural language understanding. Although only 38.4% complained about systems, that only understand single words, 57.4% voted for capabilities to understand full sentences. 79.5% even want to actively tell the computer their concern instead of answering lots of questions. Of course this ability includes the understanding of full sentences. This can be interpreted as follows: While approximately 40% regard the lack of understanding full sentences as severe, 60% think it would be good if systems could understand full sentences and 80% wish for a technology that enables the users to actively tell their story. 29.5% voted for a better synthesised voice and only 1.7% thought that there is no need at all to improve current systems.

Afterwards, we asked if speech dialogue systems are beneficial. 26.8% said yes and 44.8% think they will be beneficial in the future, given that they work better. Alarmingly, every fifth participant claims that he does not need these systems today, nor will he in the future.

6.2.3 Preferred Style (Self Assessment)

In the third part of our survey, we asked users which style they prefer. 51.0% want to use full sentences (“*How will the weather be in Paris tomorrow?*”), while 34.5% prefer medium-length commands (“*Weather in Paris for tomorrow*”). The rest voted for short commands like “*Weather, Paris, tomorrow*”. But also the system can produce sentences. Here, 63.9% want the system to formulate full sentences (“*I have found five flights. The next one, with flight number 376, leaves at 4pm, ...*”) instead of short enumerations and keywords (“*Five results: Flight 376, 4pm; flight 874, 5pm; ...*”). 57.7% prefer open-ended questions like “*How can I help you?*” to specific questions, that result in long menus (“*Do you want A, B, C or D?*”).

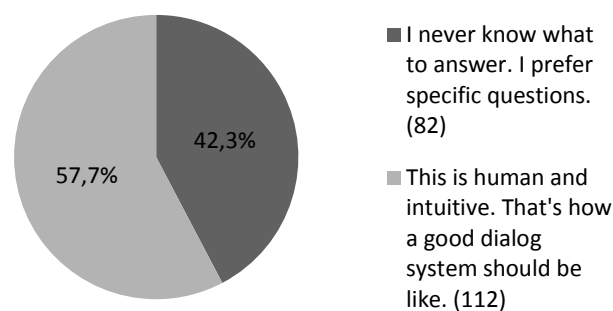


Figure 6.3: What do you think of open questions?
(Questions without predefined answer like “How may I help you?”)

About half of all participants want the system to be polite and want it to say please and thanks. 36% don’t care and only 16% think this is unnecessary. Again, slightly more than 50% are polite when addressing the system, while 36% sometimes say please and 12% don’t do it at all.

6.2.4 Preferred Style (Examples)

When addressing a computer with a command, users only use few words and seldom make use of polite formulations. This type of interaction is comparable with pressing a button to switch on the light. On the other hand, when addressing a co-worker, people mostly use very polite formulations. Edlund et al. (2008) explain this behaviour with the interface metaphor: “*Within the interface metaphor, the spoken dialogue system is perceived as a machine interface – often a computer interface. [...] Within the human metaphor, on the other hand, the computer is perceived as an interlocutor: a being with human-like conversational abilities*”.

Apart from instructions, we also wanted the participants to choose their favourite dialogue style. 71.1% preferred dialogue A in question 17 (see Figure 6.5). It is very natural and human-like, starts with an open-ended question and allows the user to control the dialogue flow. But still, every fourth user likes the restricted dialogue style more,

perhaps because they are used to it. In the last question (see Figure 6.6) the participants had to choose between three dialogue styles for getting weather information. Dialogue B was the most human-like, dialogue A was still more natural than most of today's systems and dialogue C was the machine-like style that we know from current dialogue systems. 52.6% chose dialogue B as their favourite, while 38.1% voted for dialogue A. Only less than 10% prefer the system-initiative dialogue.

6.3 Analysis

Having this data, we can now infer relations between the survey results and characteristics like age or gender.

6.3.1 Computer Usage

We first want to know if the type of computer usage affects the wish for politeness of spoken dialogue systems. We also want to make sure that the big amount of computer scientists in our study does not tamper the results. Nevertheless, we still have to be aware of the fact that only technical oriented people participate in an online survey.

Politeness: We found that 41.3% of the IT professionals say it is important, that the computer is polite (38.8% don't care) as opposed to 54.7% of the non-IT people (30.9% don't care), and 52.9% of the home users (47.1% don't care). Many IT professionals are used to command lines and seem to see the computer as a tool. Regular users may be happier with a more personal computer that acts as a partner to help them fulfilling the task.

On the other hand, 9% of the IT professionals and 15% of the non-professionals state that they are always polite when speaking with a computer. Interestingly, only 6% of the hobby users address the computer in a polite way. At the same time, this group has the highest value of people saying, that they are sometimes polite (41%).

Open-ended questions: Open-ended questions are popular among IT professionals (63%) and hobby-users (65%). But only 53% of the non-IT people, who use the computer at work like this type of questions.

Full sentences: 60% of the computer scientists want to use full sentences, whereas only 44% percent of the people who use computers at work and 47% of the hobby users want to formulate their requests in full sentences.

6.3.2 Gender

We now look for gender-specific characteristics.

Politeness: For females, politeness is more important (54%) than for males (45%).

Open-ended questions: We could identify a male preference for open-ended questions (62% vs. 53%).

Full sentences: For males it is more important (67%) that the computer uses full sentences, than for females (60%). About half (51%) of both women and men also want to be able to use full sentences. 36% of the women and 34% of the men prefer medium-length sentences and slightly more men (15%) than women (14%) think single words are sufficient.

6.3.3 Age

Because we only have 1 and 6 participants in the youngest and oldest category, we omit these in the following results and only regard ages from 19 through 65.

Politeness: Our hypothesis is that the older people are, the more do they appreciate politeness. This has not been verified, as the following results show:

- 19 – 25: 42,9%
- 26 – 35: 52,2%
- 36 – 49: 55,2%
- 50 – 65: 52,9%

Open-ended questions: We also thought that older people prefer open-ended questions, because this is more human-like. Again, there is no strong evidence for that. In fact, the favouring of open-ended questions seems to decrease with increasing age (until the age of 49). However, the age group from 50–65 has, as expected, the highest percentage.

- 19 – 25: 59,2%
- 26 – 35: 57,6%
- 36 – 49: 51,7%
- 50 – 65: 64,7%

Full sentences: Another hypothesis is that older people favour full sentences. We can see that the age group 50–65 indeed has a 7% higher preference for full sentences, while at the same time the 26–35 group likes full sentences 11% less than participants of the groups 19–25 and 36–49.

- 19 – 25: 69,4%
- 26 – 35: 57,6%
- 36 – 49: 69,0%
- 50 – 65: 76,5%

6.3.4 Analysis of the Examples

We now try to verify the answers by giving the participants examples, where they should choose the dialogue they like most. We already learned, that when it comes to actions that control devices, like switching lamps (see Figure 6.4, question 15), people mostly use short commands. Apart from the mere frequencies, we also analysed contingency tables. Only 35% of the participants who claimed always to be polite said “*please*” and 4% said “*could you please*”. Similarly, 69% of the people preferring full sentences only say “*light*” or “*light on*”.

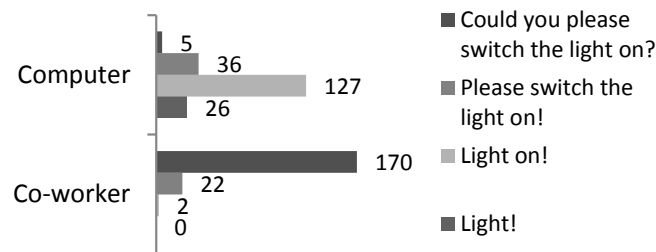


Figure 6.4: If you want to switch the light on, what do you say?

In question 17, that you can see in Figure 6.5, we presented the participants two dialogues. One being very human-like with the ability to understand full sentences, and another one being more machine-like and command-based. We analysed the results in relation to the following features.

- a) Full sentences:
 - 83% of the participants, who prefer full sentences, like dialogue A most.
 - Surprisingly, only half of the participants, who prefer short commands, like dialogue B most.
- b) Many tedious questions:
 - 72% of the participants, who don't like tedious question-answer-dialogues, prefer dialogue A. Also 70% of the people who actively want to tell the computer their concern instead of answering lots of questions prefer dialogue A.
 - Interestingly, also 70% of the participants who are not annoyed by the tedious question-answer dialogues, like dialogue A most.
- c) Open-ended questions:
 - 81% of the participants, who like open-ended questions, prefer dialogue A.
 - Surprisingly, 57% of the participants, who claimed not to like open-ended questions, still prefer dialogue A. Only 19% of the people, who like open-ended questions, prefer dialogue B.
- d) Directed dialogue:

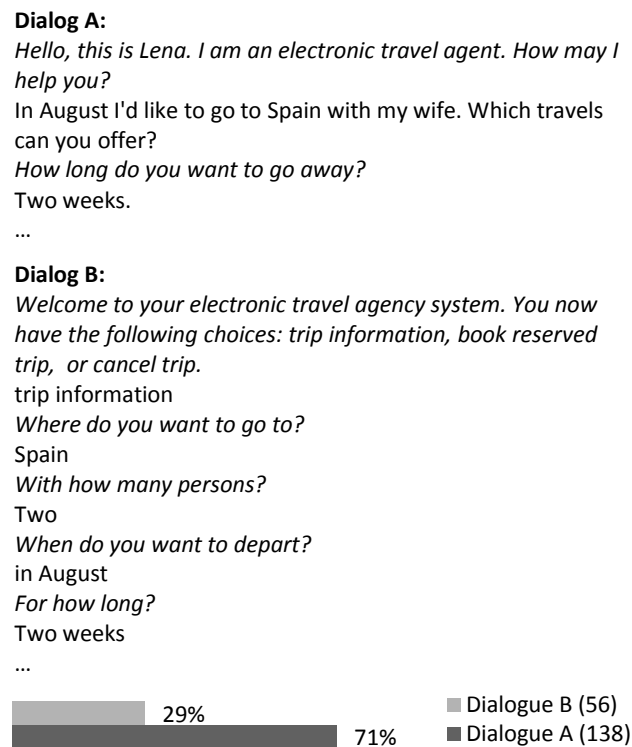


Figure 6.5: Which dialogue do you think is more pleasant?

- 75% of the participants who have problems with directed dialogues (“*Say balance, change plan, or failure report*”) or just think they are tedious prefer dialogue A.
- On the other hand, only 53% of the people, who like this dialogue-style, prefer dialogue B.

Now we ask the participants to choose the best out of three dialogues (see Figure 6.6). This time we also offer a version with medium-length commands. Again, we checked with reference to different relations:

- a) Full sentences:
 - 27% of the participants, who prefer full sentences, like dialogue A most. 66% of them prefer dialogue B (7% C).
- b) Many tedious questions:
 - 53% of the people, who don’t like systems that ask lots of questions, prefer dialogue B and 41% prefer dialogue A (6% C).

c) Open-ended questions:

- 60% of the people who prefer open-ended questions, like dialogue B and 32% prefer dialogue A (8% C).
- Interestingly, almost half of the participants (43%), who dislike open-ended questions prefer dialogue B.

Dialog A:

Welcome to your weather information service. Just tell me day and city and I'll tell you the weather forecast.

tomorrow, Paris

Tomorrow will be 25°C in Paris.

the day after tomorrow, Paris

The day after tomorrow will be 26°C in Paris.

Dialog B:

Welcome to your weather information system. How may I help you?

How will the weather be tomorrow in Paris?

Tomorrow will be 25°C in Paris.

And the day after tomorrow?

26°C

Dialog C:

Welcome to your weather information service. Please tell me the day!

August 17th

In which city?

Paris

On August 17th will be 25°C in Paris. Do you want to choose another day or a different city?

another day

Please tell me the day!

August, 18th

On August 18th will be 26°C in Paris.

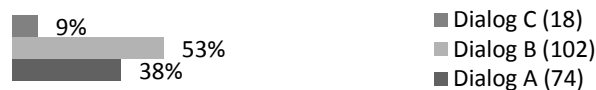


Figure 6.6: You want to know how the weather will be. Which dialogue system would you use?

6.3.5 Familiarity with Dialogue Systems

Finally, we put a special emphasis on the opinion of new users, i.e. participants who have never used a dialogue system before. We made the following observations:

- 77% prefer open-ended questions

- 72% prefer dialogue A in question 17 and 78% prefer dialogue A or B in question 18
- 72% are between 19 and 35, 67% are female, and 61% use a computer daily
- 61% say commands like “*light on*” when it comes to control situations
- Only 28% prefer directed dialogues with many question-answer-pairs

6.4 Conclusion

In this survey we have identified problems of current SDS and analysed how users would like to interact with them. 91% of all participants have already used a SDS. Interestingly, 72% stated that SDS are already beneficial today or believe that they will be beneficial in the future if they are improved. This underlines the importance of this area of research for both academia and industry. But there are still some drawbacks that need to be improved. Almost 63% do not like the tedious way of having to answer lots of questions, before being able to get the relevant information. Instead, almost 80% want to actively describe their problem. This is at odds with only 51%, who say that they would like to use full sentences and 64% who want the system to make use of full sentences. 58% prefer open-ended questions. We assume that the participants are not aware, that these features are needed in order to realise the above mentioned characteristic. Consequently, after asking self-assessment questions, we presented the participants example dialogues, to verify their claims. We found that even more people favour human-like systems, than indicated in the questions before. 71% like the most natural dialogue in question 17, which makes use of an open-ended question and full sentences. In question 18 only 9% favour the directed dialogue. Also, half of the people having said, that they only like short commands, surprisingly like the very natural dialogue most. Also 70% of the people, who claim not to be annoyed by long question-answer pairs, still prefer the natural dialogue. Even 57% of the participants, who claim not to like open-ended questions, prefer exactly this type of dialogue. Moreover, only 53% of the people that prefer directed dialogues really choose the directed dialogue as the best one. When we take a look at users who have never used a SDS before, we can as well clearly identify a preference for open questions (77% as opposed to 58%). Only 16% regard polite systems as unnecessary. The rest either favours politeness or doesn't care. This shows that polite systems are important for most people, although only 50% are polite themselves when addressing the system. This also confirms Edlund et al. (2008), who say that some people respond to greetings and social utterances, while others don't. Although anthropomorphic systems may not seem more intelligent, we can clearly see a preference for those systems, i.e. systems with open-ended questions, understanding and generation of full sentences, and the use of polite formulations. This is in line with the results of Dautenhahn et al. (2005), who found that 71% wish for a human-like communication with robots. However, we could not identify a clear relation between the wished style and age, gender, or IT skills.



Conclusions

Despite some authors claim that we do not need natural and human-like dialogue systems, we think this will be the future. It may be the case that human-like systems are the reason for difficulties because the analysis of natural language utterances is more complex than processing restricted commands like in simple command-and-control systems, but we do not believe that users wouldn't be happy with more human-like systems, if they existed. This has also been verified by our studies where most users claim that they prefer human-like dialogues.

However, in order to improve dialogue systems, we need to identify more specific requirements. With the term *natural* we refer to human-like. But a human-like formulation is not a single, static way of expressing information. It rather means to be able to adapt to the communication partner and to the situation. People are individual and have different preferences. Although about 70% prefer *natural* dialogues, this includes different styles. Moreover, the remaining 30%, who prefer machine-like, directed dialogues must not be neglected. This underlines the need for adaptive dialogue systems. But which features of a dialogue system can be adapted to make the dialogue appear more human-like? In this thesis we concentrate on formulation aspects. Some users regard the system as a friend and want to talk to it in a colloquial and informal way, whereas others see the system as an agent with whom they prefer to communicate formally. But also politeness and the length of utterances can be adapted to the user.

A severe drawback of today's dialogue IDEs is the missing flexibility for defining system utterances. Adaptive systems cannot base on a static utterance per prompt that has been defined by the dialogue designer in advance. Instead, we need language generation support. So the aspect of adaptive formulations is twofold: on the one hand we need to be able to realise them and on the other hand we need to be able to define them as abstract concepts in the development environment. This requires the identification of different utterance types, their goals and different ways to express these.

Another issue in connection with a human-like formulation is the correct classification of a user utterance. Sometimes, requests are formulated as questions or a question actually is a command. Also sub-dialogues can introduce misunderstandings. Today's systems often require the user to directly react on a question. However, in human communication, questions cannot always be answered immediately but may require more

information first. Imagine you want to go to a sunny place and the travel agent asks you where you want to go. A possible reaction could be the question “*How is the weather in Barcelona?*”. Simple keyword-oriented systems would interpret your question as an answer indicating that you wanted to go to Barcelona. Hence, the correct classification of the user utterance is a base requirement for a natural dialogue system. In summary, we found that:

- 71% prefer a human-like communication with machines
- current systems have too many options that confuse the user
- often the user does not know which category to choose
- 80% want to describe their problem actively instead of answering lots of questions
- only 16% think politeness in dialogue systems is unnecessary
- people are different: some have a menu-oriented mindset whereas others are more task-oriented
- in control situations 61% use short commands, but when the task gets more complicated, users tend to use a more complex and complete formulation
- many people are mentally conditioned when it comes to operating machines – they try to imagine a GUI
- many people use short commands because they do not expect systems to understand full sentences
- there is no clear relation between age, profession or gender and the preferred style

These two experiments answer our first question: “*What kind of problems do humans have when communicating with dialogue systems and how do people want to communicate with them in the future?*”. We can summarise that there is a clear wish for more human-like systems, although the specific style depends on the user and the situation. So we think that an adaptive formulation of the system utterances is beneficial and results in more natural dialogue systems. This requires a working dialogue model that we develop in the next chapters.

Part III

Modelling of Information-seeking Dialogue Systems

Synopsis

The modelling of a dialogue requires structural and behavioural information as well as specifications about their processing. This includes both the interpretation of user utterances and the generation of system questions. We need to know what features are required for the NLU module to interpret the answer and we also need properties that allow a natural language generation module to realise the question. The first step of interpreting the user utterance is to infer the intended action. That's why we analyse the connection between utterances and their effect on the system in Chapter 8. This allows the dialogue engine to decide what should be the next action. Once this information is known, we need a more detailed description of the system question and its possible answers. Hence, in Chapter 9 we describe an abstract question description that can be used for interpretation purposes and also contains information for the surface generation, which is presented in Chapter 10. The results build the basis for a dialogue model that describes the questions and answers in relation to each other in Chapter 11. Finally, in Chapter 12 the model is implemented and the features of the resulting system are illustrated.

Language is a part of our organism
and no less complicated than it.

(Ludwig Wittgenstein, 1889 – 1951)



Utterances and Their Effect on the System

When addressing a dialogue system, the user always has an aim. He has a certain request in mind that he wants to have fulfilled. Especially, if the system is able to handle different tasks and starts with an open-ended question, the user's intention may not be clear. In the following example the system does not understand the user's question as a request for information and just interprets *London City* as the answer to the system's question. If the system had been able to categorise the user's utterance as a question, this misunderstanding could have been prevented.

Dialogue 8.17:

S: *Please tell me your destination!*

U: *Is there a direct connection to London City?*

S-a: *Ok, your destination is London City. Where do you want to start?*

or:

S-b: *No!*

U-b: *Ok, then I'd like to go to Heathrow.*

In the next example, in both cases the user tells the system that he likes to go to Paris. However, while the first two utterances follow a regular question-answer adjacency pair, the second part of the example is special: the user does not ask a question and the system still reacts with the correct information.

Dialogue 8.18:

S: *Where do you want to go?*

U: *I'd like to go to Paris.*

S: *I can offer a 5-day trip starting in July. Would that be okay?*

U: *I'd like to go to Paris.*

S-a: *(no response because there was no question)*

or:

S-b: *I can offer you a 5-day trip starting in July. Would that be okay?*

We clearly see that the identification of the user's aim is crucial for an appropriate system reaction. This also includes the correct detection of the type of back-end access. While

the first utterance in the next example only reads information from the back-end, the second utterance changes a value.

“Is the light in the basement switched on?”
“Please switch the light in the basement on!”

In this chapter we are going to identify different classes of user utterances and their relation to the back-end. We will then sketch a dialogue system that makes use of this differentiation.

8.1 Base Elements of a Dialogue

We begin by identifying parts of a dialogue on a general level. A dialogue consists of questions and answers. We can instantly see that these categories are strongly connected with the form of an utterance. But we have observed that *form is not function*. Nevertheless, we stick to the original idea and only rename the base-elements (question and answer) of a dialogue into two more general terms: *concern* and *reply*, hoping that people do not assume a related style. This categorisation can be referred to the conversational moves of the *HCRC Map Task project: initiation* and *response* (Carletta et al., 1996). A concern comprises all types of utterances which have the aim of causing a system reaction. This can be a regular question, a command, a request or just a wish. We summarise both a command and a question under the same category as they both constitute a form of system request. A reply is any possible response which satisfies the concern, i.e. an answer or an acknowledgement. In a natural dialogue a concern does not have to be directly followed by a reply, as subdialogues and clarification questions represent a standard pattern in human communication. So together with the *speaker* of an utterance we can create four base units: user concern (UC), user reply (UR), system concern (SC) and system reply (SR). The following dialogue shows the application of these categories.

Dialogue 8.19:

SC: *Tell me your destination!*
UC: *How is the weather in San Francisco?*
SR: *It's sunny!*
UR: *Ok, then I'm heading for San Francisco.*

After analysing several dialogues, we realised that the combination of a SC and a UR equals a UC: *Tell me your destination + San Francisco = I'd like to go to San Francisco.*

$$f(SC + UR) = UC \tag{8.1}$$

For the back-end it does not matter if the request was a UC or if it was triggered by a UR in response to a SC.

8.2 Back-end-oriented Dialogue Acts

In the following sections we elaborate on the connection between the user's utterance and the corresponding result on the system. We have to remember that understanding the user's aim is learning about the function of an utterance. Recall that speech acts define what acts we commit by uttering words, i.e. they describe what a speaker *does* when he *speaks*. The actions a user can perform when interacting with a SDS clearly relate to the back-end. So in this section we describe a first draft of our back-end oriented dialogue act model which we are going to refine throughout this chapter. This model is a customised dialogue act hierarchy that is optimised for task-oriented dialogue systems. These dialogue acts are used to describe the basic goals a user has when uttering a sentence and allow the mapping between user intentions and system functionality.

8.2.1 Types of Back-end Functions

While most models start from the linguistic side (what could the user say and what consequence does it have) we specify the model bottom-up. We have a back-end and we know what it is able to do. Afterwards we determine how we can address these functions, i.e. which linguistic form triggers which function. We refer, according to McTear (2004), to three types of mixed-initiative, task-oriented dialogue systems: control systems (e.g. for controlling the lights in a smart room by speech), question-answering systems (the user asks questions in natural language), and information-seeking/booking-dialogues (i.e. the user states his goal and the system asks questions in order to gain information that is necessary to fulfil the user's request). In this connection we can identify three main categories of user aims:

- the user wants the system to realise a request (a command)
- the user wants to retrieve information from the system (a question)
- the user gives information in order to enable the system to provide him with information (an answer)

These categories relate to the basic functions each back-end provides: `do`, `getInfo` and `setInfo`, as can be seen in the following examples:

- Could you please switch the light on? → `do`
- Play some music! → `do`
- How is the weather in London? → `getInfo`
- I'd like to start on May 4th. → `setInfo`

A `getInfo`-call sends a request to the back-end, `setInfo` changes or sets the value of a variable in a frame and `do` executes a function (switch the light on, start a presentation software).

8.2.2 Selection of Dialogue Acts

Now, that we have identified our back-end functionalities, we have to relate them to the aims the user might have. This can be done with the help of dialogue acts.

When analysing existing dialogue act tagsets, we observe that they are very detailed and also include parts we excluded from our considerations, namely commitments, promises, oaths, nominations or threats. Also the recognition of turn-taking and stalling is not important to understand the intention of the user in relation to the back-end. In task-oriented dialogue systems the user has a specific goal. This goal is defined within the range of services that the system offers. Thus, we limit the dialogue acts to those ones that directly refer to the back-end. Moreover, many dialogue act schemata suffer from the fact that form is mixed with function: Both wh-questions (“*Where do you want to go?*”) and wh-requests (“*Tell me your destination!*”) have the same intention and ask for information about the destination. So from the back-end perspective it would be useful to assign the same acts to utterances that invoke the same function. However, in DIT++ the aforementioned utterances would be classified as question (information-seeking) and request (action-discussion). Our approach bases on Bunt’s second-level general-purpose functions (Bunt et al., 2010). We select *information seeking functions*, *information providing functions*, *commissives* and *directives* and introduce some changes. On the processing side of our dialogue systems we don’t need commissives, as we do not concentrate on human-human-like conversations. Of course the system could produce utterances like “*I will look for that*”, but from the back-end processing perspective (at this point) we don’t need to understand promises, invitations, oaths or threats. From the *directives* we only use the *instructions*-category and rename it into *action requesting* in order to delimit from the form again (instructions are often associated with imperatives but also a question can be used to instruct somebody). These dialogue acts can now be related to our former mentioned back-end functions (see also in Figures 8.1 and 8.2):

- an *information-seeking* dialogue act will initiate the `getInfo` function,
- an *information-providing* act initiates the `setInfo` function, and
- an *action-requesting* act calls a `do` function.

The first two methods are information-related while the last one refers to all other actions, like switching devices. We must not mix up these categories. An instruction like “*Search for hotels in London that offer breakfast*” is not an action request but an information seeking act because it refers to the retrieval of information instead of the execution of an external action that results in an alteration of data.

Apart from these acts, we also have to do with what in DIT++ is called *social obligations*, like greeting/return greeting. These acts often don’t need any back-end access, which means that they bypass it. Moreover, they often form symmetric adjacency pairs, so the reaction belongs to the same dialogue-act-category as the request. Hence we name them *copy* dialogue acts. Most of these are part of the social/conversational domain. But not all of the social obligations fall within the copy category, as e.g. an *introduction*

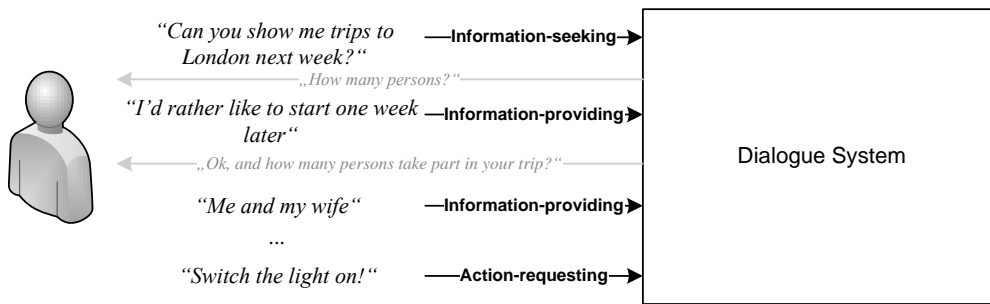


Figure 8.1: Back-end-oriented dialogue acts

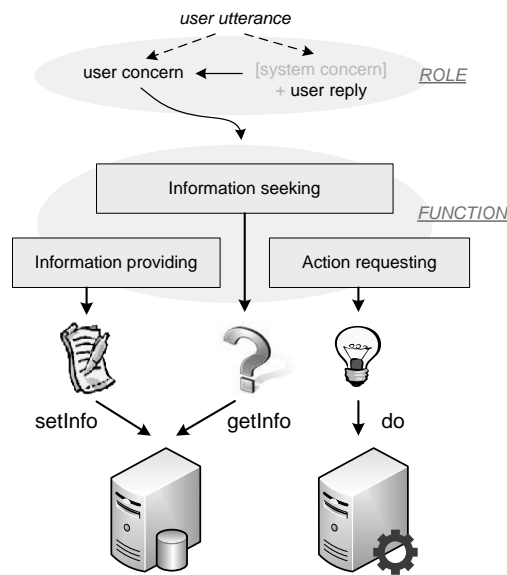


Figure 8.2: Back-end-oriented dialogue acts and related functions

requires back-end access for further usage (the system has to memorise the user's name). The self-introduction, for example, equals the goal to get the partner's name (`getInfo` in the social domain). Of course this does not always reflect an actual interest in the name, but the social obligation requires a "returned introduction". Regardless of the underlying motivation, from the system's point of view an introduction is an information seeking act (like "What's your name?").

8.3 Comparison with the CRUD-Approach

We now compare *dialogue acts* with the database-oriented *CRUD approach*. This comparison views dialogue systems from two different perspectives: Linguistics and Conceptual Modelling.

8.3.1 The CRUD Approach

We have introduced basic back-end functions and their relation to classes of human utterances. This is important in order to address the correct function of the back-end and is a necessity for fulfilling the user's goal. But the addressing of back-ends is not only an issue in language processing. Especially in the context of information systems and databases it has to be dealt with. A well established description or generalisation of the system's abilities can be found with the acronym *CRUD*, which refers to the functions *create*, *read*, *update* and *delete*. These functions are closely related to the basic SQL commands *insert*, *select*, *update* and *delete*. Möller (2013) describes CRUD as "*the four fundamental atomic operations common to persistent database systems*". They "*are all individual, low-level operations, rather than phases in a sequential lifecycle*". Moreover, he states that "*the term is now also widely used to describe the functionality of applications at a higher, even user interface level*".

8.3.2 CRUD in Natural Language Utterances

When designing speech interfaces, we have to deal with the connection of the user level and the actual back-end functions as well. So this classification not necessarily refers only to databases. We are interested in the question how CRUD relates to dialogue systems and to dialogue acts. The simple assumption that information seeking acts refer to select operations, information providing acts refer to update/insert operations and action requesting acts refer to create/delete/drop operations is not valid as we will explain hereafter.

In many speech dialogue systems, frames are used to aggregate information. Recall, that a frame is an object-oriented approach for collecting and storing data as a (hierarchical) set of key-value pairs. The keys represent the information demand of the system whereas the values represent the user's answers. Every entry of a frame is called slot and its manipulation refers to the change of the value of a specific key. The filling or manipulation of a frame's slot can also be regarded as an update- or insert-operation. Here, we have to differentiate between operations on the frame and operations on the actual back-end. Although we are able to relate CRUD to all frame-based operations, it gets ambivalent when also being applied to the actual back-end request.

So we clearly have to distinguish between a dialogue act (that should relate to the whole dialogue system) and operations on either the frame or the back-end, as you can see in Figure 8.3. In information systems, most tasks focus on getting existing information. Other task-oriented systems also include data manipulation. This is (seldom) the input of new data with the help of Natural Language Interfaces to Databases and (more often)

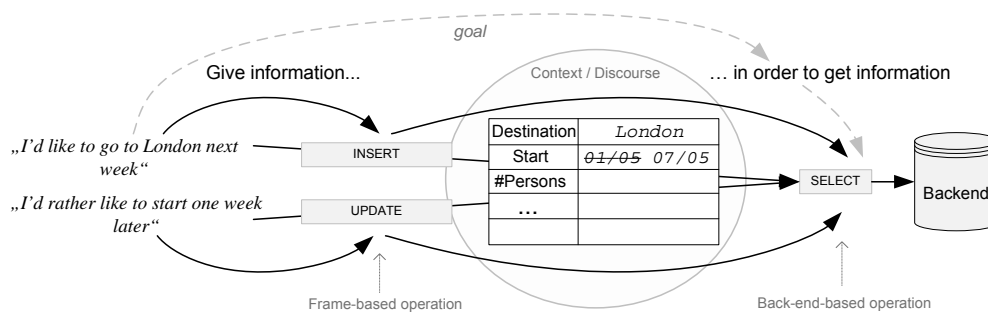


Figure 8.3: CRUD

the manipulation of states of objects (mostly data models), like switching on the light, setting the volume level to 60% or even to book a trip or to make a reservation. Given that our main functions are *information seeking* and *action requesting*, we clearly see that an information seeking request – like “Which trips to London do you have next week?” or “I’d like to go to London next week” – includes an update or insertion of the dialogue context (specification of location and time and consequently the update of the frame). Nevertheless, the overall goal is a select-operation on the back-end (retrieve all trips that are available under the given terms).

At this point, we see that dialogue acts relate to the user’s intention or final goal whereas CRUD refers to the type of operation on any data model. In the case of dialogue systems we have to deal with two different data models: the frame (as a temporary short time storage to collect the user’s goal in several dialogue steps) and the back-end (that provides the actual information that the user is interested in). We have seen that an information seeking act is connected with the user’s goal to gather information and includes an insert-operation on the frame and a read/retrieve-operation on the back-end (select). When we take a look at action requesting acts, we encounter a similar characteristic. Their goal is the execution of a process that manipulates a model (this can be data in the back-end or the change of states of physical objects). They can also include information that triggers an insert-operation on the frame. But in contrast to information-seeking acts, an update-, insert- or delete operation will be executed on the back-end, as we can see in the following examples:

“Please switch the desk lamp on!”
 “Set the volume to 60%”

In this case, of course, the given information is the basis for creating the condition. Here, we can indeed observe a data manipulation in the back-end (`update devices set state=true where devID=365`). But what about information providing acts? One might be tempted to argue that they relate to update operations. But again we have to specify if we refer to the frame or to the back-end. Information that the user provides is (in

the context of task-oriented dialogue systems) meant to update the dialogue context (i.e. frame-related). It does not change the goal (e.g. to book a trip vs. asking for weather information vs. switching a lamp on) and it is not intended to result in a database update (“*I’d rather like to go next week*” only influences the select condition but not the data). Consequently, information-providing acts only refer to an insert- or update-operation on the frame and do not refer to the back-end at all, as we can see in the summary in Table 8.1.

act	CRUD (frame)	CRUD (back-end)
inf.seek.	(insert/update)	select
act.req.	(insert/update)	insert/update/delete
inf.prov.	insert/update	-

Table 8.1: Dialogue Acts vs CRUD

8.4 System-oriented Dialogue Acts

By comparing the CRUD approach with our preliminary dialogue act model that we called *back-end-oriented dialogue acts* (BoDA) we found out that we have to differentiate three layers: (a) frame-related, (b) back-end-related, and (c) the combination of both, i.e. a generalised system layer which is represented by a dialogue act. While for the user there is no difference between them, as a developer we have to separate these layers in order to make the system-internal relationship between CRUD and dialogue acts unambiguous.

In Table 8.1 we can see that the notion of information seeking and action requesting dialogue acts matches the back-end related CRUD approach (information seeking: select data, action requesting: manipulate data), whereas all kinds of BoDA result in an insertion or update of the frame. An information providing act does not define a different goal. It is just an update of the frame in the current context. We can also argue, that both information seeking and action requesting acts entail an information providing act. This would make a dedicated information providing act superfluous. But since natural language utterances exist, that are only meant to introduce new or update existing parameters of the search query without changing the goal, this qualifies for an own category. The following examples show dialogue acts with entailed information provision and standalone information providing dialogue acts.

- (A1) Sys: “*When do you want to go to London?*”
- (A2.1) User: “*How is the weather next week?*”
(information within information seeking act, change of goal)
- (A2.2) User: “*I’d like to go next week*”
(standalone information providing, answer)

or:

(B1) U: “*I’d like to go to London next week*”
 (information within an information-seeking act)

or:

(C1) S: “*When do you want to go to London?*”
 (C2) U: “*In two weeks*”
 (C3) S: “*And do you prefer a train connection or a flight?*”
 (C4) U: “*I’d rather like to go next week*”
 (standalone information providing, correction of previous information, no change of goal)

Utterance A2.1 is an information seeking act because it changes the goal. A2.2 is an answer and thus a standalone information providing act. B1 looks very similar to A2.2 but the former is an information seeking act because it introduces a goal and is just another way of saying “*What trips do you have for London next week?*”. The utterance represents an indirect search request and resembles a classical SQL-statement with a condition (`select * from trips where destination='London' and departure='01/07/2012'`). However, “*next week*” clearly is information providing. In this case the information seeking act entails an information providing act. By this means it triggers a frame-related insert- and a back-end-related select-statement.

The statement C4 has no effect on the back-end at all. The resulting frame update constitutes only an a posteriori change of the select condition. This is possible because the frame can be seen as a buffer for information that needs to be collected before a database request can be formulated. A standalone information providing act is always an answer. It only contains new attributes in the context of an existing goal. An entailed information providing act can also be seen as an a priori answer to a presumed question, as can be seen in the following example:

(U1a) *I’d like to book a trip.* (inf.seek)
 (S2a) *Where do you want to start?*
 (U3a) *London* (inf.prov)
 =
 (U1b) *I’d like to book a trip to London*
 (inf.seek. with entailed inf.prov.)

In this case, the user’s concern is not complete, which makes the system ask for the missing information. So U3a is a supplement to U1a.

As a consequence of the fact that information providing acts refer to the frame and not to the back-end, we rename *Back-end oriented Dialogue Acts* more generally into *System-oriented Dialogue Acts* (SoDA). They regard the complete dialogue system as a black box

and describe only what the user expects from the whole system. The internal frame- and back-end-related operations can be described by CRUD. So the main categories of SoDA are *information seeking* and *action requesting* acts. Both types result in a frame-related update or insertion. The difference can only be seen when regarding the underlying operation on the back-end: information-seeking acts result in a select statement, whereas action-requesting acts result in a change of data, i.e. an update-, delete-, or insert-statement. Moreover, both can include an *information providing* act. However, also autonomous information providing acts are possible, mostly in the form of answers. In order to distinguish an information seeking act from an information providing act, we need to analyse the context. In the following dialogue there are two possible user answers. Both are information providing acts and therefore set information in the frame, i.e. they set the attribute *city* to *London*.

Dialogue 8.20:

S: *Where do you want to go?*

U-a: *I want to go to London.*

U-b: *London.*

However, the formulation of the first answer can also be used in a different role, as we can see in the next example. This time the user has the initiative and does not answer a question, yet he still gives information about the destination. Consequently, his utterance has to be classified as a concern or more specifically we can say that his intention is to get information about trips to London. Hence, we can categorise this utterance as *information-seeking*. It has the same result as the command “*Show me trips to London*” or the question “*Do you have trips to London?*”.

Dialogue 8.21:

U: *I want to go to London.*

S: ...

As we already mentioned, we want to delimit from the form or style of the utterance and instead only focus on the effect on the system. This brings us to the main differentiation between information-providing and information-seeking acts. An information-providing act only includes values for attributes but never changes the task (i.e. the selection of the correct frame). Hence, we need to know about the current task in order to detect if a given task really is a new one. This is also a reason why speech acts (instead of dialogue acts) are not sufficient. For the correct identification of the user’s intention we always need to be aware of the discourse. The importance of the task as main feature for distinguishing the dialogue acts can also be seen in the following example:

Dialogue 8.22:

S: *Where do you want to go?*

U-a: *I want to go to London.*

U-b: *I want check the weather in Paris first.*

While the first answer does not change the task (trip booking), the second answer introduces a new one (check weather). So the first answer is a standalone information-

providing act and the second one is an information seeking act (get weather information) that encloses an information-providing act (Paris). Another way of explaining the difference between those acts is the observation that the answering of a question invokes some action on the system – may it be the change of a value in the frame or an operation on the back-end – in the same way as a command would do. In the first case the system may implicitly define a task by asking for the destination. By providing this information, the user confirms that he has the same goal and thus creates a concern in which he wants to get information about trips to London. This means that the combination of a system concern and a user reply has the same effect as a user concern, which also confirms our initial assumption.

8.4.1 Elements of an Utterance

We have realised that information seeking and action requesting acts may include information providing acts and that we can tell apart these acts by the existence or absence of a task. An act describes the user’s intention on a very general level and can be specified in more detail by:

- domain
- task / action
- attribute
- attribute value

In the following annotated example we can see the different elements of an utterance.

“I’d like to [travel]_{domain} [to]_{attribute} [Cologne]_{attr.value} [tomorrow]_{attr.value}.”
act=information seeking, domain=travel booking, (task=get trips)

“I’d like to [make a reservation]_{task} for [train]_{attribute} [6754]_{attr.value}.”
act=action requesting, (domain=train booking)

Every user has a goal that he wants to have fulfilled. This can be done by executing the correct task (i.e. the related action) and requires a certain amount of information that is specified by the attributes and the corresponding values, which together form a frame like this:

$\left[\begin{array}{l} \textit{destination} : \textit{Cologne} \\ \textit{date} : \textit{tomorrow} \end{array} \right] \quad \left[\textit{train} : 6754 \right]$

Attributes can be part of any system-oriented dialogue act, whereas the task can only be specified by information seeking and action requesting acts. Information providing acts have no effect on the task and can be part of the first mentioned acts. A task is always associated with a specific domain and related to an action that the user wants to execute.

8.4.2 Description of Dialogue Utterances

Every utterance is uttered by a *speaker* s and can be associated with an *act* A that describes the speaker's primary illocution, i.e. the intention the user has when uttering his concern. This intention is independent from the *form* f , since an utterance that is formulated in an interrogative style does not necessarily need to be an information seeking act (“*Could you close the door, please?*”). So, due to their ambiguity (stylistic or functional scope), terms like *question* or *command* do not qualify as a proper description of the utterance. The *role* r of an utterance can be derived from the act. Information-seeking and action-requesting acts are *concerns* and information-providing acts are *replies*. An utterance is further described by a *task* t that it tries to accomplish. Every task is associated with a *domain* d and is characterised by a certain information demand that can be addressed in the utterance by reference to *attributes* a and their values. This leads to $U = (s, r, A, d, t, a, f)$. But since the role can be derived from the act, the result is the following sextuple:

$$U = (s, A, d, t, a, f) \quad (8.2)$$

where

$$\begin{aligned} s &\in \{user, system\} \\ A &\in \{inf.seeking, inf.prov., action req.\} \\ d &\in \{domain1, \dots, domainN, dialogue, social\} \\ t &\in \{task1, \dots, taskN\} \\ a &= \{(attribute1, value1), \dots, (attributeN, valueN)\} \\ f &= (sentence\ type^1, mode^2, style^3, \dots) \end{aligned}$$

and

$$r = \begin{cases} reply & \text{if } A = \text{inf.prov.} \\ concern & \text{else} \end{cases}$$

Now that we have defined the features, we will use them to annotate some examples: The utterance “*I’d like to go to San Francisco*” may be interpreted as a wish, a request or just a statement. For the back-end this makes no difference as it is only important to know what reaction the user expects from the system. In our approach we describe this

¹declarative sentence, question, ...

²indicative, subjunctive, imperative

³formal, informal, ...

utterance as follows:

$$U = (\text{user},$$

$$\text{information seeking},$$

$$\text{travel},$$

$$\text{get trips},$$

$$\{(destination, SanFrancisco)\},$$

$$(\text{declarative sentence}, \text{subjunctive}))$$

This can be read as an *information seeking user concern in the travel domain with the aim to get trips to San Francisco* that would result in a `getInfo` back-end call. However, depending on the context, the utterance could also be an answer to the question “*Where do you want to go?*”, which would change the act to *information providing*. As already mentioned, this differentiation can only be made under consideration of the discourse by detecting a change of the task. The sentence “*Could you please close the window?*” can be described as:

$$U = (\text{user},$$

$$\text{action requesting},$$

$$\text{smart room},$$

$$\text{switch},$$

$$\{(window, off)\},$$

$$(\text{question}, \text{subjunctive}, \text{formal}))$$

This would result in a `do` back-end call. The question “*How is the weather in Paris?*” leads to:

$$U = (\text{user},$$

$$\text{information seeking},$$

$$\text{weather},$$

$$\text{getForecast},$$

$$\{(city, Paris)\},$$

$$(\text{question}, \text{indicative}))$$

Finally, the utterance “*hello*” is expressed as:

$$U = (\text{user},$$

$$\text{copy},$$

$$\text{social},$$

$$\text{greet},$$

$$\emptyset,$$

$$(\text{single word}, \text{informal}))$$

System-oriented dialogue acts generalise the user’s intention with respect to the dialogue system. This enables us to recognise “*How will the weather be like in Munich tomorrow?*” as a question instead of an answer to the question “*What is your destination?*”, though being able to use basic contextualised keyword spotting mechanisms. Also “*Could you please switch the lights off?*” or “*Could you tell me if the lights in the basement are on?*” can be correctly classified as an action requesting or information seeking utterance. This differentiation is crucial for the dialogue manager for being able to decide how to respond to a request and what action to take.

With the more detailed description introduced in this section, we are not only able to categorise utterances but also create a surface independent representation of the user’s utterance that is the basis for language understanding and the execution of the correct system action.

8.4.3 Recognition of System-oriented Dialogue Acts

The first step towards the creation of an utterance description is the identification of the dialogue act. This is vital for the dialogue manager and the inference of the correct back-end function, because a question might refer to a `getInfo`- or to a `do`-request as the following example illustrates:

“*Would you please shut down the projector?*” \leadsto `do`
“*Would you please tell me when the next bus leaves?*” \leadsto `getInfo`

There is no difference in form, but nonetheless both questions lead to different back-end functions. The reason for this is strongly connected with the two levels of illocution. The primary illocution is the actual *goal* of an utterance while the secondary illocution refers to the *form* of the utterance. In the first example, the secondary illocution is a question whereas the primary illocution is an action requesting act.

We now briefly sketch an approach on how to recognise dialogue acts. Please note that this is not the primary goal of the thesis. We only want to prove the viability of the SODA tag set instead of finding the best performing algorithm. For an overview of dialogue act recognition techniques, including an N-gram approach, Hidden Markov Models, Bayes classifiers and neural networks, please refer to Fishel (2007).

There are several characteristics that can be used to describe utterances and infer corresponding dialogue acts. Fishel (2006) identifies linguistic, prosodic, keyword-based and statistical features. We will concentrate on linguistic and word-based features, i.e. sentence structures, parts of speech and cue phrases. Webb (2010) describes cue phrases as “*single words, or combinations of words in phrases, that can serve as reliable indicators of some discourse function*”. Although there is no one-to-one relationship between form and function, there might exist syntactic structures that indicate a certain primary illocution. So the combination of lexical cues and syntactic characteristics may lead to a useful heuristic.

Webb refers to a list of 62 cue words identified by Hirschberg and Litman (1993), some of which you can see in Table 8.2. We observe that these words do not relate to our three top-level dialogue acts. The relation between the cue phrase *hello there* and the

accordingly	first	moreover	similarly
again	further	namely	still
alright	furthermore	next	so
also	gee	nevertheless	then
alternately	hence	nonetheless	therefore

Table 8.2: Cue words

dialogue act *conventional opening*¹ is obvious, but the relation between more complicated utterances and our system-oriented dialogue acts is more challenging. So we have to find cues and syntactic structures that indicate *information seeking*, *information providing* and *action requesting* dialogue acts.

In order to find potential cue phrases, we have to create a corpus, identify n-grams with a relative frequency greater than a certain threshold and finally calculate their confidence in relation to the dialogue acts. This does not work in our worst-case scenario from the beginning of this section, as the potential cue phrase “*would you please*” may have a high relative frequency but at the same time has low confidences for all dialogue acts, because it matches different dialogue acts (action requesting and information seeking). So we have to concentrate on the difference in the utterances, e.g. the type of the verb. An utterance with the verb *shut* leads to a change of the state of a real object (do) whilst *tell* refers to the collection of information (`getInfo`). So we have to differentiate between three types of verbs according to the dialogue acts (see Table 8.3). However, general verbs like *want* or *like* do not indicate specific dialogue acts, because they can be used in different contexts: “*I’d like to go to Paris*” vs. “*I’d like to know the extension of my consultant*”.

inf. seeking	inf. providing	action requesting
say		switch
tell me		turn
know		close
name		open
...		...

Table 8.3: Cue verbs

The basis for our approach will be Table 8.4. As you can see, both information seeking and action requesting utterances have similar features. So the verb category, along with the occurrence of a question-word, is the most eminent marker for the type of dialogue act. Because there are only few attributes that indicate information providing dialogue acts, they need to be identified by exclusion or, as mentioned in the last section, by

¹example taken from Webb (2010)

analysing if the utterance changes the task.

<i>class</i>	<i>instance</i>	inf. seeking	inf. prov.	act. req.
mood	indicative	×	×	×
	conditional	×		×
	imperative	×		×
	interrogative	×		×
verb category (<i>cue verbs</i>)	information gathering	×		
	action executing			×
question word	Wh	×		

Table 8.4: Potential features for recognising system-oriented dialogue acts

We now want to strengthen the assumption that verb cues essentially support dialogue act recognition. According to Table 8.4, the sentence “*Could you please turn the light on?*” can only be correctly identified as *action requesting* by the verb *turn*. So we now apply some of these features to a machine learning algorithm. We decided to use a Maximum Entropy Classifier, because it does not require independent attributes (as for example a Naive Bayes Classifier does). This classifier bases on conditional probabilities, i.e. we predict a class c by the given data d (sometimes also called *evidence* or *observation*). Because we are only interested in the most probable class we just return the class with the maximum likelihood:

$$\arg \max_{c \in C} P(c | d) \quad (8.3)$$

$$C = \{\text{information seeking, information providing, action requesting}\}$$

The data is represented as a feature vector that is extracted from the user utterance. We select the following features:

- conditional
- interrogative
- information_seeking_verb
- action_requesting_verb
- no_cue_verb
- no_cue_verb_and_no_wh_word
- wh_word
- no_wh_word
- short_utterance

As you can see, we also add the lack of a characteristic as a feature (to enable the algorithm to find conclusions like *in case there is neither an information seeking verb nor an action requesting verb, chances are high that we deal with an information providing act*), as well as combinations of features, which makes them dependent (hence we should not use a Naive Bayes Classifier). The features are represented as indicator functions f

that take as input a class and the data (Manning and Klein, 2003).

$$f_i(c, d) \equiv [c = c_i \wedge \Phi(d)] \quad (8.4)$$

e.g. $f_1(c, d) \equiv [c = \text{inf.seek.} \wedge \text{wh_word}(d)]$

This function indicates if the feature occurs in the data d and for which class c it is a marker. The result is a boolean value. Instead of only calculating relative frequencies (cf. Bayes Theorem: $P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)}$), the maximum entropy classifier assigns weights λ to the features and adapts them in an iterative process, which is known as *multinomial logistic regression*. We now train the classifier with 15 utterances (see Appendix on page 204). Afterwards we can compute the probability for the given data and learned weights to belong to a certain class (Manning and Klein, 2003).

$$P(c|d, \lambda) = \frac{\exp\left(\sum_i \lambda_i f_i(c, d)\right)}{\sum_{c' \in C} \exp\left(\sum_i \lambda_i f_i(c', d)\right)} \quad (8.5)$$

e.g. $P(\text{inf.seek.} | \text{'When do you want to depart?'}, \lambda) = \frac{e^{\lambda_1 + \lambda_3}}{e^{\lambda_1 + \lambda_3} + e^{\lambda_x} + e^{\lambda_y + \lambda_z}}$

given that f_1 and f_3 are true for d .

After the learning process (100 iterations), we predict the dialogue acts for 13 test utterances, which results in a precision of 100%. When removing the verb cue features we only get 38%, which supports the importance of verb cues. In the latter case, even a test on the training set gives us only 73%. However, we must be aware of the fact that this experiment bases only on small training and test sets so that results in productive use may differ.

8.4.4 Application in a Dialogue System

As we now have a way for describing the function of an utterance, we can integrate this outcome into a dialogue system model, which uses the dialogue acts as foundation for further processing.

In Figure 8.4 you can see a refined version of our first dialogue system black box model from Figure 3.1 on page 39. After the speech recogniser has translated the user utterance into text, it is sent to the natural language understanding component. The first step in the understanding process identifies the user's aim. So we perform a dialogue act recognition, which allows us to tell apart a *concern* like “*How is the weather in London?*” from a *reply* like “*I want to go to London.*”. This is essential to address the correct back-end function. In order to classify the user utterance, we first convert it into a feature vector (we use the identified features from the last section) and then apply this vector to a machine learning algorithm that has been trained beforehand. However, if an utterance has been classified as *information providing*, a plausibility check, that we

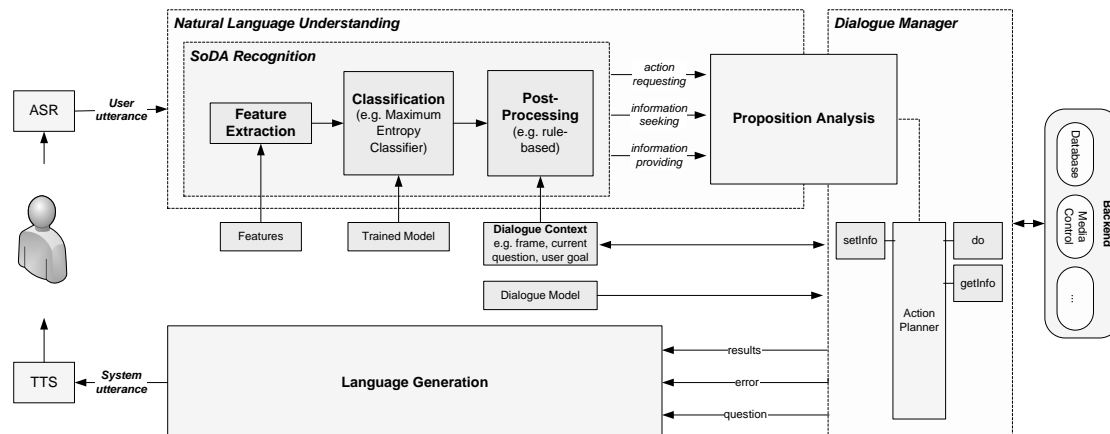


Figure 8.4: Dialogue system

realise as a rule-based post-processor, is necessary because an information providing act only makes sense if we have an active, unanswered question in the dialogue context and if the user utterance contains an answer to this very question. Otherwise the result will be changed to *information seeking* or *action requesting* respectively. This is necessary to correctly identify ambiguous utterances like “*I’d like to go to Munich*”, as can be seen in the following example:

Dialogue 8.23:

U: *I’d like to go to Munich.* → *information-seeking (user initiative)*

vs.

S: *Where do you want to go?*

U: *I’d like to go to Munich.* → *information providing*

vs.

S: *Do you like a room with sea view?*

U: *I’d like to go to Munich.* → *information seeking (correction)*

The check whether the user utterance answers an unanswered system question is only a simpler way to find out if a new task has been introduced by the user, which is a signal for an information-seeking or action-requesting act, or in other words, excludes an information-providing act as a solution. This check overcomes potential classification errors that may have been introduced by the sparse amount of features that point to an information-providing act.

One consideration is to add the facts whether an unanswered question is active and whether the user utterance matches the question as features for the machine learning algorithm. However, this approach is not recommendable. Remember that we want to recognise the user utterance U1 in the following example as information seeking, in order to not consider it as a candidate answer to the question.

Dialogue 8.24:

S: *Where do you want to go?*

U1: *How is the weather in Munich?*

U2: *I want to go to Munich.*

However, the feature `utterance_matches_question` would be true for both U1 and U2, provided we use keyword spotting. Consequently, it is useless for their distinction. So the question if the user utterance matches the system question is only relevant if it previously has been classified as information-providing act and you want to check if it should rather be an information-seeking act instead. This argumentation also applies to the feature `unanswered_question`. It can only be used to exclude the class information-providing from the range of solutions. The feature itself has no power to indicate if an utterance actually is information-providing (an unanswered question does not mean that the following user utterance is an information-providing act as you can see in Dialogue 8.24). Hence, we put the evaluation of these features into a post-processing step.

When the system has identified the dialogue act, in the next step the actual proposition of the utterance needs to be extracted (at this point we treat this as a black box). This component requires information from the dialogue context and dialogue model and thus sits at the border between language understanding and dialogue management. Knowing the type of act helps to select the correct element from the dialogue and prevents choosing an element that alters information when the user asks a question. Depending on the type of act, the dialogue manager chooses the corresponding function (`getInfo`, `setInfo`, `do`) to address the back-end (e.g. database, media control (Crestron, AMX), presentation control). The parameters are then filled by the results from the proposition analysis. In case of an information-providing concern the information is just stored in a frame without triggering any back-end-access. Depending on the result, the dialogue manager decides to ask a further question, present the results or notify the user about an error. The first case applies if the command is ambiguous or if the user has not given all the required information, as you can see in the following example:

Dialogue 8.25:

U: *Turn on the light!*

S: *Do you want to turn on the ceiling- or the desk lamp?*

...

U: *Could you recommend a city trip for next weekend?*

S: *For how many persons?*

Finally, the information that should be presented to the user is transmitted to the language generation component that produces an appropriate sentence. This sentence is then converted into speech by the speech synthesis module.

8.5 Summary

In this chapter we have described the relation between user utterances and the back-end. We first took a look at the base elements of a dialogue and the relation between question and answer. We then identified three classes of back-end access (getInfo, setInfo, do) that are the basis for a back-end oriented dialogue act definition. The subsequent comparison with the CRUD approach enabled us to establish two views or layers of the dialogue system (frame and back-end) which resulted in a refinement of our model. This led to *System-oriented Dialogue Acts* and an abstract, surface-independent description of user utterances that includes the act, speaker, domain, task, attributes and syntactical features. These acts are necessary to remove ambiguity from user utterances without the necessity for a deep understanding of the sentence and help to bridge the gap between current industry applications and research prototypes. As a proof of concept we selected features for these acts and trained a maximum-entropy model that successfully identifies SoDA mostly by the use of lexical cues. Finally, we refined our dialogue system overview and sketched the application of SoDA in a dialogue system.

Language is the means of getting an idea from my brain into yours without surgery.

(Mark Amidon)



Categorisation of Questions and Their Answers

In this chapter we focus on information-seeking acts and their replies. We already learned that dialogue acts can be used to describe utterances in a very general way. However, they are neither sufficient to model questions on a level that allows us to process the user's answer nor do they provide us with information on how to generate system questions. With the existing model we cannot categorise the type of the question in more detail, i.e. what type of information the question asks about and what type of information the answer to this question should provide.

We believe that the classification of questions is crucial for different parts of a dialogue system and is especially important for Rapid Application Development purposes because the manual definition of prompts as well as the formalisation of the answer domain is a time-consuming and recurring task. A common taxonomy of question types will help to connect questions with the natural language understanding (parsers, grammars) and language generation modules (grammar or pattern-based). This refers to two use cases:

- generate a system's question of a specific type
- process the user's answer to this question

Thus, in this section we present an overview of different question type classifications and then propose an abstract question description. However, it is not intended as an approach to understand user's questions, although parts of the description may also be helpful for that.

9.1 Possibilities of Question Classification

Most question taxonomies refer to the area of tutoring systems. A well known taxonomy by Bloom et al. (1956) consists of the elements *comprehension*, *application*, *analysis*, *synthesis* and *evaluation* and clearly refers to teaching. Also Boyer et al. (2009) refer to tutorial dialogue and distinguish 29 types of questions. This includes tutorial question types like *calculation* or *definition* and more general ones like *clarification* and *quantification*. From the Information Retrieval and Question-Answering point of view, there

are two main classification schemes: In the *Message Understanding Conference* (Chinchor, 1997) questions are classified as *named entities* (person, organisation, location), *temporal expressions* (time, date), and *number expressions* (money, percent), whereas in the *Text Retrieval Conference* (Voorhees, 2003) *factoid*, *list*, and *definition* questions are distinguished.

According to Hirschman and Gaizauskas (2001), questions can be distinguished by answer type (factual, opinion, summary) and by question type¹ (yes/no, wh, indirect request, command). The last two categories describe utterances like “*I would like to know . . .*” and “*Name all persons that . . .*”. When using the term *question* we mean all information-seeking utterances and explicitly also include these last two examples and do not only refer to utterances with a trailing question mark. An often encountered problem in connection with command-like formulated questions is the missing wh-word. So instead of saying “*Where do you want to go?*” a possible formulation would be “*Name your destination!*”. Thus, we need to find a general way of classifying interrogatives. So we now analyse existing question type schemata.

Classification by Category:

Probably the most prominent classification has been done by Graesser et al. (2008) who define the following 16 categories:

- Verification
- Disjunctive
- Concept completion
- Feature specification
- Quantification
- Example
- Definition
- Comparison
- Interpretation
- Causal antecedent
- Causal consequence
- Goal orientation
- Instrumental/procedural
- Enablement
- Expectation
- Judgmental

Many of them refer to tutoring and do not relate to task-oriented dialogue systems. Instead of asking informative questions like asking for the number of persons or travel dates, they ask for expectations or judgements or ask questions about the content. Apart from asking for new information, also existing information needs to be checked (verification, clarification). When looking at Graesser’s classification again, we can see that only the first five categories seem to be relevant for task-oriented dialogue systems. Concept completion refers to *wh-questions*, feature specification refers to questions that expect an adjective as answer, and quantification questions ask for numbers. However, these

¹also called *kind of question* in Hirschman and Gaizauskas’s work

types are not expressive enough, neither for inferring parsers nor for generating questions. Moreover, *disjunctive* questions don't qualify for an own category in this context as they only influence the surface and not the intention or answer domain of the question. Indeed, most questions can be formulated in a disjunctive way, e.g. “*Where do you want to go?*” and “*Do you want to go to Paris or London?*” both refer to a location, although the latter is more restrictive. Also a *list-question* as defined in TREC (Voorhees, 2003) does not change the range of a question. In fact, one can't be sure how many answers exist for a question. “*Tell me the names of all countries beginning with Ö*” may lead to only one answer, whereas “*Who invented the phone?*” may lead to several people. As you can see, Graesser's specification mixes purpose (verification), style (disjunctive) and basic types (quantification).

Classification by Question Words:

A very basic classification can be done by question words like *who*, *what*, *where*, *etc.* This leads to interpretation difficulties because there is no link to the answer type. A what-question can for example refer to a time (“*What's the time?*”) or a building (“*What is the highest building in the world?*”).

Classification by Question Words and Answer Type:

Consequently, in order to specify concept completion, Srihari and Li (1999) define classes for factoid questions, i.e. named entities, temporal expressions, and number expressions. They extend the named entity classes from the Message Understanding Conferences by types like duration, weight, area, email, temperature and many more. This provides “*a better foundation for defining multiple relationships between the identified entities*”. Moreover, they introduce the *asking point* of a question, i.e. more or less the question word (sometimes in connection with an adjective) like *who*, *why* or *how long* which relates to a specific answer type (e.g. person, reason, length).

Also Moldovan et al. (1999) realise that the question word is not sufficient for finding answers because question words like *what* are ambiguous. So they decide to use a combination of question class (question word), subclass and answer type, e.g. what/what-who/person (“*What costume designer decided that Michael Jackson should only wear one glove?*”), what/what-where/location (“*What is the capital of Latvia?*”) or how/how-many/number (“*How many people live in China?*”). Here the *question-focus*, i.e. the word the question is looking for, is the key for the correct identification of subclass and answer type (e.g. focus: capital, answer type: location).

Classification by Answer Type:

We clearly see that the answer type plays an important role when classifying questions. Hamblin confirms that by putting up the following postulates (Stokhof and Groenendijk, 1994):

- An answer to a question is a sentence, or statement.

- The possible answers to a question form an exhaustive set of mutually exclusive possibilities.
- To know the meaning of a question is to know what counts as an answer to that question.

So defining the answer helps defining the question. Existing work describes different levels of answer types. Whereas Hirschman and Gaizauskas (2001) distinguish *factual*, *opinion* and *summary*, Moldovan et al. (1999) define categories like *money*, *number*, *person*, *organization*, *distance* and *date*. Also Srihari and Li (1999) introduce similar types that they call *named entity classes*.

Classification by Intention:

Also Hovy et al. state that “*there are many ways to ask the same thing*” (Hovy et al., 2000). So they analysed 17,384 questions and answers in order to create a QA typology for Webclopedia. They did not focus on question words or the semantic type of the answer but instead attempted to represent the user’s intention. So instead of classifying “*Who was Christoph Columbus?*” as who-question or “*What are the Black Hills known for?*” as what-question, they are both tagged as *Q-WHY-FAMOUS*.

Classification by Dialogue Act:

In *DIT++*, questions are defined as *information seeking functions*, that are divided into *set questions*, *propositional questions* and *choice questions*. A set question is described as wh-question, propositional questions are yes/no-questions and choice questions can be answered with one element of a list of alternative choices. This classification resembles a shallow answer type classification but is too superficial for the task at hand. Also SoDA are too abstract because their intention is the description of the general user goal (towards the whole system) whereas we need to find a way to represent the actual proposition of the question. Consequently, dialogue acts need to be refined by a more granular specification.

9.2 Abstract Question Description

Questions are strongly connected to answers: A who-question results in an answer that contains a person. So the question word *who* and the answer type *person* are self-evidently closely related. Still, the answer type is not bijectively bound to the question word. Also the question word *what* can relate to the answer type *person*. So the classification of the following questions is not easy. They all have the same aim, yet the formulation is fundamentally different.

- “*Who was the first man on the moon?*”
- “*What’s the name of the first man on moon?*”

- “I’d like to know the name of the first man on the moon.”
- “Name the first man on moon!”
- “Do you know who was the first man on the moon?”

The first question can be classified as *factual question* (Hirschman and Gaizauskas), *concept-completion question* (Greasser), as a question that asks for a *named entity* (Srihari and Li), as a question that asks for a *person* (Moldovan), as a question with *focus* “first man” (Moldovan), or as a question with the *asking point* “who” (Srihari and Li). As you can see, there are many different possibilities how to classify these questions. Recall that the question type mostly refers to the formulation and cannot be used to group questions according to the information they ask for. Moreover, because a question is a specific type of dialogue act, we also need to differentiate between secondary and primary illocution. The last question looks like a yes/no- or enablement-question but in fact should be answered with a name. This also reinforces the problem of classifying a question only by structure or question word. Instead, for task-oriented systems, the user goal is most important, i.e. what kind of information the user expects.

We now propose an Abstract Question Description² (AQD) that can be used to refine information seeking SoDAs. It describes questions from different perspectives in order to satisfy different use cases such as classifying by intention and classifying by style. Hence, we differentiate type and form.

9.2.1 Type

The type describes the data we expect in the answer. It can be used to classify questions with the same intention but with a different formulation as the same category. Moreover, we can specify parsers for the natural language understanding module according to the available types.

Answer Type

In order to understand and process a question, the expected answer type is the most important information, because it allows to classify questions depending on the type of information they ask for. The combination of different classification schemes leads to a taxonomy with increasing specificity, e.g. `fact > named entity > animated > person > astronaut`, `fact > named entity > non-animated > location > city`, or `decision > yesNo`. We differentiate six classes (decision, fact, open ended, description, reason, opinion) that split up in many types and subtypes. However, we focus on the first three classes with special attention drawn to *facts*. An overview can be seen in Figure 9.1.

Apart from classes and their subtypes, we also consider *constraints*. Constraints are type restrictions. These are boolean functions that check if a given value satisfies certain conditions, e.g. if a speed is in a certain range, if a city name is in a given set, or if the date is smaller than today. These conditions are specified as boolean formulas (e.g.

²AQD 2.0; this is a revised version of the original AQD that was proposed in Berg et al. (2012a)

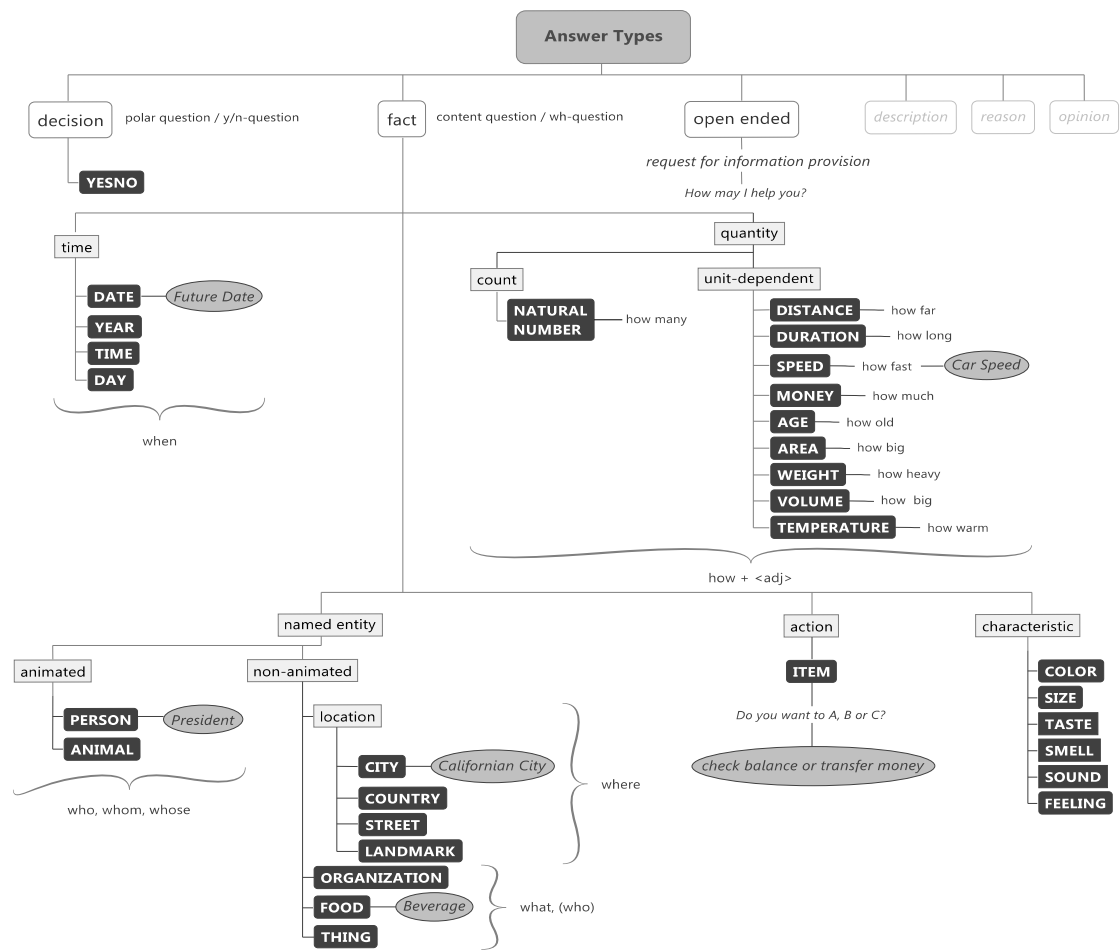


Figure 9.1: Answer Types

conjunctive normal form) and indicated in curly brackets with x referring to the data to classify, e.g. $Speed\{(x>0) \wedge (x<101)\}$ or $City\{x \in (London, Paris, Berlin)\}$. As an abbreviation, type restrictions can be denoted in square brackets for ranges and round brackets for sets, e.g. $Speed[1..100]$ or $City(London, Paris, Berlin)$. As a further simplification, these constraints can also be referred to by named links to the actual constraints, e.g. $City\{\$EuropeanCapitalCity\}$ may link to $City\{x \in (London, Paris, Berlin)\}$. So, we can imagine further constraints like $City\{\$CalifornianCity\}$ or $Person\{\$President\}$.

When taking a closer look at the following formulations, that all have the same intention, we can see that answer types provide an abstract generalisation of the question.

1. Where do you want to go to?

2. What's your destination?
3. Do you already have a destination in mind?
4. I need to know where you want to go.
5. Tell me about your travel plans!

In this example the type could be `fact.namedEntity.nonAnimated.location.city`.

Reference Type

Because a dialogue act describes what someone *does*, it is closely related to verbs (to ask, to inform, to command). When refining information-seeking acts, we describe what we ask for, e.g. to ask for a location. By this means we describe the answer type of a question, e.g. location or distance. In this connection we observe that a question may refer to different types at the same time. The question “*Do you want to go to London?*” obviously is a decision question that should be answered with yes or no. At the same time the question relates to a location (city). So “*No, to Paris*” would be a valid answer, too. Likewise, “*Would you like a drink?*” can be answered with “*A coke please*”, i.e. without mentioning *yes* or *no* at all. This phenomenon is closely related to immediate correction and over-informative answers and thus mostly occurs in connection with polar questions, like verifications or proposals.

In order to address this very natural feature, we distinguish between *answer type* and *reference type*. The former refers to the type that is expected (direct answer type), whereas the latter specifies a category that the question refers to (indirect answer type). As well as the answer type, the reference type can also be specified in more detail by constraints.

Cardinality

In the Text Retrieval Conferences, questions are divided into the categories *factoid*, *list* and *definition*. This classification is unfavourable for our issue as it mixes answer type class (factoid, definition) and cardinality (list). Also, as you can see in the following examples, the expected cardinality of the answer does not always match the actual cardinality (notation: assumption/fact, e.g. N/1).

- “*Which players were in the German Soccer Team of 2012?*” → N/N
- “*Who wrote ‘Do Androids Dream of Electric Sheep?’?*” → 1/1
- “*Who produced ‘Avatar’?*” → 1/N
- “*Name all international capitals beginning with Z!*” → N/1
- “*Name the editor of ‘i, Robot!’*” → 1/N

This is due to the fact that the interrogator does not know the answer in advance and makes an assumption in order to be able to formulate the question. Although the interrogator uses a plural in the fourth question, there exists only one city. So in many cases it is not possible to infer the number of answers from the formulation of the question. Furthermore, we have to differentiate between the cardinality itself and the actual value of an answer, i.e. a *how many*-question only has a single answer (cardinality=1). In our opinion, the cardinality is negligible for the categorisation of questions. But as this feature may be important for question generation and can also be helpful to enable the dialogue manager to only return as many answers as the user wants to have, we still include the assumed cardinality into our description. This allows the distinction of the following two questions:

```
“Name five players of the Scottish Rugby Team!” →  
  type: fact.namedEntity.animated.person{$$ScottishRugbyPlayer}  
  cardinality: 5  
“Can you name a player of the Scottish Rugby Team?” →  
  type: fact.namedEntity.animated.person{$$ScottishRugbyPlayer}  
  cardinality: 1
```

9.2.2 Form

The question (“*When do you want to leave for London?*”) and the request (“*Tell me when you want to leave for London!*”) both have the same intention, i.e. getting the time of departure. When classifying by intention, both interrogatives should be of the same type. This demand has been fulfilled by the type-level of our AQD. For question generation, however, we need more information about the style. While the classification by question types has been declared impractical for describing the intention of an interrogative because different question words can refer to the same goal, e.g. *when* and *at what time*, it is still important for style variation.

Question Type

Instead of only considering the answer type, we need to include the question type as well. The question type refers to the style of the utterance, which refers to the chosen question word and grammatical mood. Hence, apart from the mere question word, we can also vary grammatical characteristics. When generating a question for the AQD answer type `fact.temporal.date`, we can think of different formulations:

Question (Y/N): *Do you want to go?*

Wh-Question: *When do you want to go?*

Wh-Request: *[Please] Tell me / specify when you want to go!*

NP-Request: *[Please] Tell me your departure time [please]!*

C-Wh-Question: *Can/Could you [please] tell me when you want to go?*

C-NP-Question: *Can/Could you [please] tell me your departure time?*

Command (NP): *Your departure time?*

Command (N): *Departure time...?*

We clearly see that all these utterances have the same intention and thus the same answer type. However, the existing AQD is not sufficient for a successful generation of questions because it lacks a specification of the style. We can conclude that the answer type generalises questions with reference to the type of information they ask for, whereas the question type specifies a certain formulation that we wish for.

Language

The language has no influence on the meaning of an utterance, but it plays an important role for the realisation of multilingual dialogue systems.

Surface Modifier

Some authors categorise questions by the feature *disjunctive* (or *list-question*). We consider this as a *surface-modifier* because it does not change the message of a question. In fact, most questions can be reformulated to be disjunctive: “*Do you want to go to London?*” and “*Do you want to go to London or Paris?*” both refer to the same answer- and question type, i.e. the disjunctive formulation is a surface-oriented feature. Moreover, there also exist disjunctive formulated questions that involve several answer types. In the request “*Please say your zip code or your city name!*” there are two answer types. This can be modelled by a set of answer types: {*zip, city*}. However, decision questions (i.e. yes/no questions) should not be formulated in a disjunctive way because this complicates the interpretation of the answer: “*Do you want to have a GPS or don't you want to have a GPS?*”. Further features that only influence the style of an utterance are *politeness* and *formality*, which we describe in more detail in Section 10.1.4.

9.2.3 Context

Under the *context* level of the AQD we subsume pragmatic information like the *purpose* of a question and its *role*, i.e. a more detailed specification of the meaning. The context would also include *domain* and *task*, but as these are already part of the general utterance description they are not mentioned at this point again.

Purpose

The *purpose* of a question refers to the issue why we ask a question. We might want *to check*, *to gather information*, *to control the dialogue flow* or *to fulfil social obligations*. The first category refers to questions that check existing information. We can distinguish *verifications* that ask for the truth of a given fact (“*So you want to go to San Francisco, right?*”) and *clarifications* that offer the choice between two alternatives (“*Did you say Boston or Houston?*”). The second category aims at retrieving new information that is

crucial for the processing of the task, the third category includes questions that ask for the next step in the dialogue (e.g. task selectors like “*Do you want to book a trip or do you need weather information?*”), and finally there exist social obligations like “*How are you?*” that are only important to simulate a friendly and human-like system.

Role

At this point we can describe the question by means of its form and the type of the answer. But we also need to describe its meaning, i.e. when asking for a date we need to specify it in more detail in order to understand its pragmatics. We call this the *role* of a type. Examples include birth dates or departure dates. So the role of the type `fact.time.date{x > today}` could be *return date*. However, in order to be more generic and domain-independent, we divide the role into *pragmatic aspect* and *reference* so that we can generalise the context of a return date with the pragmatic aspect `end` and the reference `trip`, i.e. a date that needs to be interpreted with respect to the end of a trip. This allows us to have different representations for the following two questions:

```
“When do you want to leave?” →  
  type: fact.time.date{x > today}  
  role: start of trip  
“When do you want to come back?” →  
  type: fact.time.date{x > today}  
  role: end of trip
```

In this way, the role describes the type that is being asked for on a pragmatic level. This very abstract description may sound strange in some cases, but this is no drawback for functionality:

```
“When were you born?” →  
  type: fact.time.date  
  role: begin of life/person  
“When was this house built?” →  
  type: fact.time.date  
  role: begin of house
```

Apart from a temporal usage, the aspects *start* and *end* can also be applied to local verbs:

```
“Where do you want to go?” →  
  type: fact.namedEntity.nonAnimated.location.city  
  role: end of trip
```

The distinction between a temporal or local meaning is provided by the type. Further examples for pragmatic aspects are *possession* and *want* as the following examples demonstrate:

```
“Do you want breakfast?” →  
  type: decision.yesNo  
  role: want of breakfast  
“Do you have a customer card?” →  
  type: decision.yesNo  
  role: possession of card
```

In combination with the answer type, the role serves as a simple description of the question’s meaning. By the differentiation into pragmatic aspect and reference we get a generic combination of basic features, rather than extremely domain-specific role descriptions like *birthday* or *departure date*.

9.3 Application of the AQD

In the last sections we have described many different aspects that contribute to the description of a question that we call *Abstract Question Description* (AQD). This description is divided into three logical layers. The *type*-layer describes a question by the type of the expected answer. This includes the description of a type that the question might indirectly refer to, which also reflects the primary and secondary illocution of a question. The cardinality describes the expected number of elements of the answer and allows us to represent list-questions. The second level of the AQD describes the *form* of the question. These features have no influence on the meaning. The form can be specified by the type of the question (e.g. wh-question), the language, and surface modifiers (disjunctive, level of politeness). The last part of the AQD is about the *context* of the question and describes it on a pragmatic level. The purpose indicates why a question is asked and the role specifies the type with regard to its use or goal, i.e. a date may be used in the context of a birthday. This results in the following description hierarchy:

- Type
 - Answer Type & Constraint
 - Reference Type & Constraint
 - Cardinality
- Form
 - Question Type
 - Language
 - Surface Modifier
- Context
 - Purpose
 - Role

- * Pragmatic Aspect
- * Reference

In general, we can describe the question “*When do you want to leave for London?*” as a question for a date (type), formulated as a wh-question (form) that refers to the start of a trip (context). In detail we would specify it as depicted in Figure 9.2.

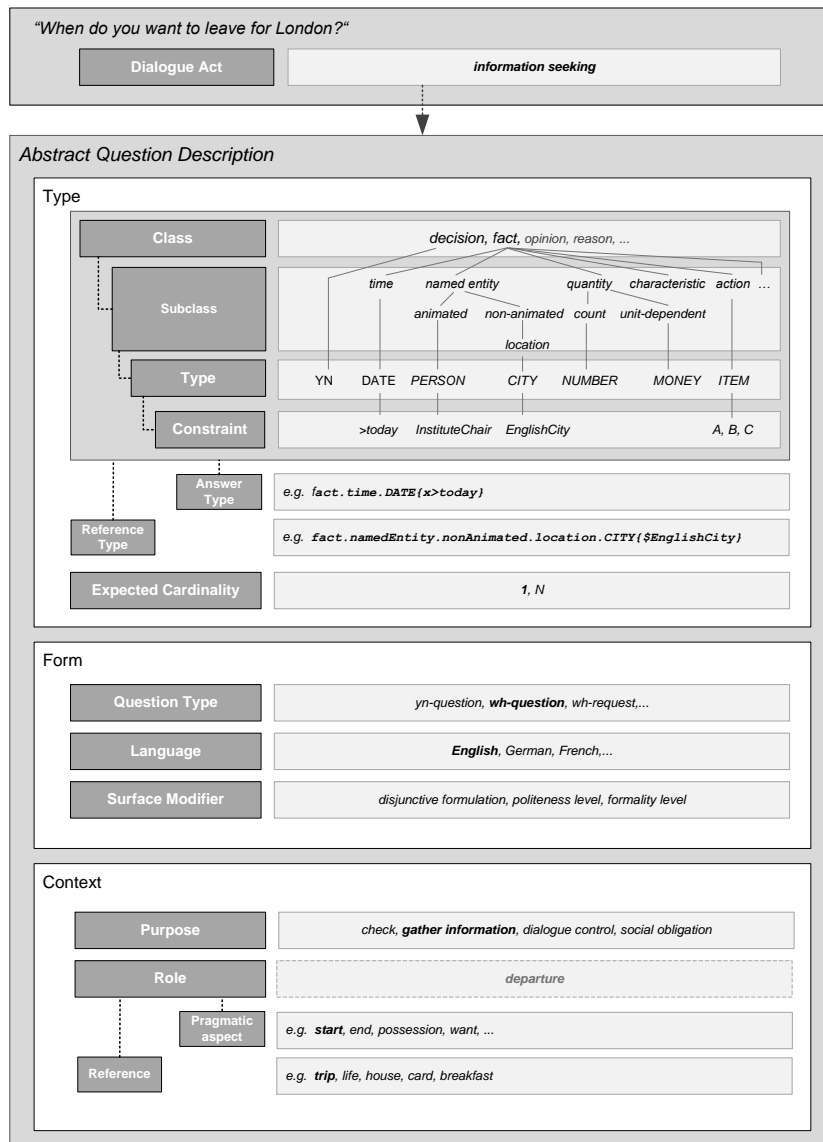


Figure 9.2: Abstract Question Description

9.4 Summary

In this chapter we have developed a description that supports the processing of user answers and the generation of system questions as a refinement of information seeking SoDAs. We have analysed various approaches to the classification of questions but did not find a question description that matches our requirements. Many existing schemata only focus on a special point of view like tutoring systems. Others lack structure and mix aspects (e.g. Graesser et al.: verification question, disjunctive question and feature specification question). An interesting structural division that is also the foundation for our model has been proposed by Hirschman and Gaizauskas, who differentiate between question kind and answer type. Whereas the question kind refers to the style of a question (e.g. wh-question or request), the answer type refers to the type of information of the answer (e.g. location). Moreover, our model bases on Hamblin's realisation that the understanding of questions strongly relates to the identification of possible answers. Consequently, we use the answer type for the classification and understanding of the answer, and we use question types for the specification of the surface form that we want to realise. The top level of our model, the *abstract question description* (AQD), is divided into three aspects:

- type: what the question asks for
- form: how the question is asked
- context: what the question is about

So, different parts of the model are helpful for different tasks. When the system asks a question, the type can be used to choose the correct grammar(s) for the speech recogniser and it defines which type of answer is expected, which is very important for the natural language understanding process and the selection of the correct parser. When the system should generate a question, type, form and context are needed to create a suitable formulation.

10

Generation of System Interrogatives

Some of the major problems Bringert (2008) identifies with current interactive speech applications is that they are not natural and not cheap enough. In our user studies we found that 71% of users indeed prefer the most natural dialogues when choosing from three fictional human-machine dialogues. This is in line with the results of Dautenhahn et al. (2005), who found that also 71% of people wish for a human-like communication with robots. Looi and See (2012) describe the stereotype of human-robot dialogue as being monotonous and inhuman. They argue that the engagement between human and robot can be improved by implementing politeness maxims, i.e. connecting with humans emotionally through a polite social dialogue. Again, we can confirm this by the results of our user study, with only 16% of the survey participants regarding polite systems as unnecessary.

Natural dialogues require a user-oriented formulation and behaviour. While some people prefer telegraphic sentences or even single words like “*Destination?*”, others expect full sentences like “*Which city do you want to fly to?*”. In many languages you also have to obey T-V-distinction¹, which refers to respectful or familiar formulation. While common systems use a predefined formulation that is only synthesised by the system, we aim at a concept-to-speech-component that automatically generates language depending on the user’s preferences.

In order to realise user-friendly and natural dialogue systems that apply politeness maxims and adapt their style to the user’s language, we need support from a language generation component. The manual definition of prompts as fixed texts in different styles would be a very time consuming, inflexible and expensive task. This has also been brought up by Bringert (2008), who says that there exist applications that even don’t make use of today’s possibilities because of the high cost of implementing such systems.

Hence, in this chapter we want to address these problems by language generation methods. We propose a way to automatically realise system questions in different styles from abstract representations. This makes dialogue systems appear more natural without leading to an extremely high effort as a manual prompt definition in different styles would cause.

¹lat. tu/vos

10.1 Natural Language Generation

Reiter and Dale (2000) describe the process of language generation as *goal-driven communication* that can be seen as “*an attempt to satisfy some communicative goal or communicative intention on part of the speaker*”. This communicative goal may be to inform the hearer, request him to do something or to obtain information from him. Reiter and Dale define the input (e.g. a call to generate an utterance in a dialogue system) to a natural language generation (NLG) system as a four-tuple $\langle k, c, u, d \rangle$. The knowledge source k stores application-dependent information about the domain, the communicative goal c describes the purpose of the text (in our case the goal of a system utterance, e.g. `ask_departure_date`) to be generated, the user model u describes the characteristics of the hearer and the discourse history d stores the text (in our case the different turns of the dialogue) that has already been generated. According to Reiter and Dale (2000), the generation process can be divided into three stages with several subtasks:

- document planning² (content determination and document structuring)
- microplanning³ (lexicalisation, generating referring expressions, aggregation)
- surface realisation (linguistic and structure realisation)

The *document planner* produces “*a specification of the text’s content and structure, using domain and application knowledge about what information is appropriate for the specified communicative goal, user model, and so forth*” (Reiter and Dale, 2000). This step defines what to say and how it should be grouped and ordered. The *microplanner* then decides how to express the intended message, i.e. which words to use (e.g. *the user*, *she*, *Lisa* or *the young woman*) and which syntactic form to choose (e.g. tense or mood). This is called lexicalisation and requires “*some mechanism for choosing amongst the alternatives*” because “*there are numerous ways in which a piece of information can be expressed*”. Additionally, in this step we aggregate sentences and generate referring expressions. The result is still an abstract representation and not yet actual text. In the *surface realisation* step, this representation is converted into text by applying grammar rules, inflecting verbs, adding auxiliary verbs and building the correct tense. Furthermore, in this step mark-up tags like paragraphs or pronunciation information (e.g. SSML⁴) may be added.

The overall process can be realised as a pipeline as depicted in Figure 10.1. The output of one component is at the same time the input for the next component. *Document plans* specify structural information of the document and consist of *messages* that describe the content on a conceptual level that can be mapped into linguistic forms. Hence, “*the notion of a message forms a useful abstraction that mediates between the data structures used in the underlying application and the eventual texts to be generated*”. Every message corresponds to a linguistic fragment that may be bigger than a sentence and refers to a certain information that the system wants to convey. This may be a message about the

²also text planning

³also sentence planning

⁴W3C Speech Synthesis Markup Language

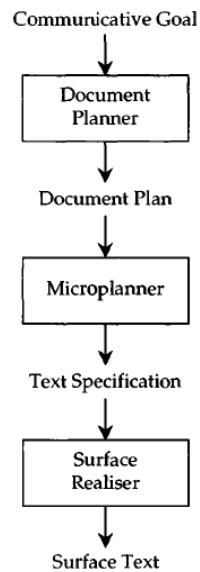


Figure 10.1: NLG system architecture (Reiter and Dale, 2000)

monthly temperature in a weather information system, as you can see in Figure 10.2. A number of messages in a certain structure (i.e. the document plan) now serves as input for the microplanner, which transforms the plan into text specifications by applying lexicalisation and aggregation methods.

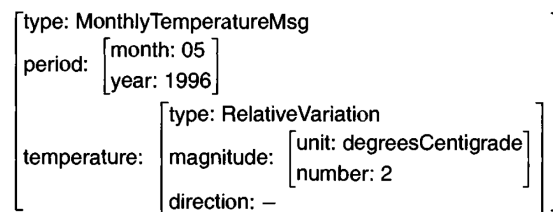


Figure 10.2: A message about the monthly temperature (Reiter and Dale, 2000)

Text specifications describe the structure of the text (e.g. paragraphs) and consist of *phrase specifications* (that describe the sentences) that we can define with different approaches. The simplest form is orthographic strings or canned text, i.e. just static, predefined text, that does not make use of high level language generation methods. An *Abstract Syntactic Structure*, on the other hand, specifies base lexemes and syntactic relations, as illustrated in Figure 10.3. Inflection and word order is handled later by the realiser. This allows changes in the surface text by just changing some parameters, e.g. changing the grammatical number to singular would result in the sentence “*The month had a rainy day*”.

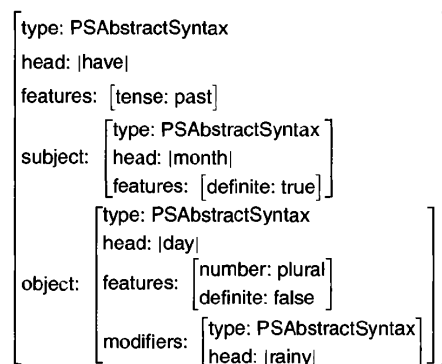


Figure 10.3: Abstract syntactic structure for ‘The month had some rainy days’ (Reiter and Dale, 2000)

When we replace the syntactic roles in Figure 10.3 like *subject* and *object* by semantic roles like *possessor* and *possessed*, we speak of *Lexicalised Case Frames*. The text specification, in what form so ever, is the input to the realiser, which requires a grammar to convert this representation into actual text.

10.1.1 Combinatory Categorial Grammar

We now want to briefly introduce a special type of grammar that is widely used to generate language. Mark Steedman’s Combinatory Categorial Grammar (CCG) formalism contains no phrase structure rules and consists only of lexical entries. Steedman and Baldridge (2011) describe this as “a form of lexicalized grammar in which the application of syntactic rules is entirely conditioned on the syntactic type, or category, of their inputs”. These categories “may be either atomic elements or (curried) functions which specify the canonical linear direction in which they seek their arguments” (Baldridge and Kruijff, 2003). Atomic elements (sometimes called saturated), such as N, NP, PP or S (the start symbol S does not need to be a full sentence), do not need to be combined with other elements. They are complete in themselves. With the help of the slash operator we can combine saturated and unsaturated elements to new unsaturated elements. The slash indicates on which side the arguments should appear. This leads to forward (argument on the right) and backward (argument on the left) application (see Equation (10.1)). We call unsaturated elements functions because a complex category like Y/X denotes an element that looks for an X on the right in order to become a Y. This is exactly the behaviour of a function that takes X as an argument and returns Y as result.

$$\frac{Y/X \quad X}{Y} > \qquad \frac{X \quad Y \backslash X}{Y} < \qquad (10.1)$$

Apart from application combinators, there are composition and type-raising combinators. Composition refers to the reduction of two complex functions to an easier one and is described with the operator B together with the application direction. This can be thought of as cancelling the first function’s argument with the second function’s result category as the following equation illustrates:

$$\frac{Y/X \quad X/Z}{Y/Z} \rightarrow \mathbf{B} \tag{10.2}$$

Type-raising turns “arguments into functions over functions-over-such-arguments” (Steedman and Baldridge, 2011), i.e. argument X is turned into a function that has a complex argument that takes this X as an argument. It allows “arguments to compose with the verbs that seek them” and is also used in order to be able to apply all rules into one direction (e.g. a full left-to-right proof). Type-raising is denoted with a T .

$$\frac{X}{Y/(Y \setminus X)} \rightarrow \mathbf{T} \tag{10.3}$$

Given the following lexicon,

<i>science</i>	⊢		NP
	<i>I</i>	⊢	NP
	<i>like</i>	⊢	$(S \setminus NP) / NP$

we can derive the sentence “*I like science*” as follows:

$$\frac{\frac{\frac{i}{np} \quad \frac{like}{s \setminus np / np} \quad \frac{science}{np}}{s \setminus np} \rightarrow}{s} \leftarrow \tag{10.4}$$

With type-raising and composition we can also achieve a full left-to-right proof:

$$\frac{\frac{\frac{\frac{i}{np} \quad \frac{like}{s \setminus np / np} \quad \frac{science}{np}}{s \setminus np} \rightarrow \mathbf{T}}{s / s \setminus np} \rightarrow \mathbf{B}}{s} \rightarrow \tag{10.5}$$

OpenCCG⁵ is a collection of natural language processing tools, which provide parsing and realisation support based on the CCG formalism. A set of XML-based files allows us to define the lexicon and the categories. For a deeper introduction into the OpenCCG syntax you may refer to Kruijff et al. (2005).

10.1.2 Question Generation

Question Generation is a twofold area of research. On the one hand, it deals with text understanding and the generation of questions related to the content. This area is of special interest for tutoring system researchers, where the system has to understand the content of an article, identify relevant sentences and formulate questions about them in order to automatically create reading assessments. On the other hand, question generation can be seen as a subdomain of language generation and *concept-to-speech* technology. Especially in rapid application development, the specification of concepts along with surface parameters instead of hard-coding system prompts can be very time saving, reduces development effort, and also introduces the possibility to adapt system utterances with regard to the demands of the user. Consequently, question generation can be divided into (a) question generation about a given text and (b) question generation from abstract concepts (semantic representations).

Based on the contributions to the *Workshops on Question Generation* (2009; 2010), most work has been done in the area of tutorial dialogue, i.e. question generation from text. Papasalouros et al. (2008) describe how to generate multiple choice questions for online examination systems, Ou et al. (2008) show how to extract predictive questions from an ontology that might be asked by the user of a question-answering system, and Olney et al. (2012) describe an approach to generate questions by filling templates. Their approach also focusses on tutorial dialogue and is based on psychological theories that claim that questions are generated from a concept map. But as the use of representations has been avoided in state of the art question answering systems, Olney et al. discuss whether a representation-free approach that bases on syntax transformations (e.g. wh-fronting) of given declarative sentences can successfully generate questions. They found that this approach has difficulties with determining the question type and conclude that a certain degree of knowledge representation is necessary. This representation is being used to fill assessment question templates like:

- Why/How did <character><verb><complement>?
- What happens <temporal-expression>?
- Why was/were <character><past-participle>?
- Why <auxiliary-verb><x>?

However, we concentrate on the rapid development of information-seeking dialogue systems and are therefore interested in question generation from semantic representations.

⁵<http://sourceforge.net/projects/openccg/>

10.1.3 Linguistic Style

Mairesse (2008) explains linguistic style as “*a specific point within the space of all possible linguistic variation*” and concludes that it can be considered as a “*temporary characteristic of a single speaker*”. He refers to Brown et al. (1987) and the need for self-esteem and respect from others and states “*that the use of politeness is dependent on the social distance and the difference of power between conversational partners, as well as on the threat of the speaker’s communicative act towards the hearer*”. Gupta et al. (2007) state that “*Politeness is an integral part of human language variation, e.g. consider the difference in the pragmatic effect of realizing the same communicative goal with either ‘Get me a glass of water mate!’ or ‘I wonder if I could possibly have some water please?’*”. Jong et al. (2008) claim that language alignment happens not only at the syntactic level, but also at the level of linguistic style. They consider linguistic style variations as an important factor to give virtual agents a personality and make them appear more socially intelligent. Also Raskutti and Zukerman (1997) found that naturally occurring information-seeking dialogues include social segments like greeting and closing. Consequently, Jong et al. (2008) describe an alignment model that adapts to the user’s level of politeness and formality, which we have already described in Section 4.3. Recall, that the model has three dimensions: politeness, formality and T-V distinction. Whereas politeness is associated with sentence structures, formality is dependent on the choice of words.

Style variation is the generation of different formulations with the same goal. The question (“*When do you want to leave for London?*”) and the request (“*Tell me when you want to leave for London!*”) both have the same intention, i.e. getting the time of departure.

10.1.4 Telling Apart Politeness and Formality

We have seen different realisations of the same message. But how do these formulations affect us and how is this related to politeness and formality? We have already learned that there exist different degrees of politeness and that systems with an appropriate choice of style are important to the user. Also, although only 50% are polite themselves when addressing the system, 71% want the dialogue system to be polite. This confirms Edlund et al. (2008), who say that some people respond to greetings and social utterances, while others do not. However, it is hard to tell what exactly makes an utterance more polite or more formal than another. Moreover, formality and politeness are very close terms that often get mixed, which is also confirmed by Jong et al. (2008). Furthermore, they state that formality is a matter of personal taste.

Often a question (“*When do you want to go?*”) seems more polite than a request (“*Tell me when you want to go!*”). But what if we add *please* to the request? Is the request more polite because of this modal particle? Is “*can you please*” more polite than “*could you*”? In this thesis we work with the following hypothesis. Politeness is characterised by:

- the use of *please*

- the use of a subjunctive modal verb
- interrogative style (question vs. request)

Formality is sometimes also regarded as influencing politeness, i.e. a very formal style is intended to be polite. In most cases this is true, but nevertheless we have to distinguish politeness and formality in order to be able to adapt to it independently. We regard formality as the lexicalisation, i.e. the choice of words and the choice of word types. In the following example we can identify different levels of formality in relation to the choice of words, i.e. *specify* sounds more formal than *tell* and *set off* sounds more colloquial than *leave*.

“Can you tell me when you like to set off?”

“Can you tell me when you want to leave?”

“Can you tell me your departure date?”

“Can you specify when you want to leave?”

“Can you specify your departure date?”

Moreover, the grammatical structure can also influence formality. While the first two sentences use a verb phrase (*you want to leave*), the third one makes use of a noun phrase (*your departure date*). Some languages (e.g. German or French) also differentiate between formal and informal personal pronouns (second person). This is called T-V distinction and affects, as an indicator for social distance, formality. Again, the exact ordering of utterances regarding formality is subject for a user study. We work with the hypothesis that formality can be influenced by:

- the choice of words
- the grammatical form of the sentence (NP vs. VP)
- the choice between formal and informal personal second person pronouns

With this clear distinction between politeness (the use of *please* and subjunctive forms, choice of question style) and formality (choice of words and grammar, T-V distinction) we have now parameters at hand to change the style of a system interrogative.

10.2 Concept-to-Text

In the last sections we have introduced the overall language generation process, the difference between politeness and formality as well as the CCG formalism. We now sketch a process for automatically generating system questions in SDS, i.e. we want to provide the development environment with information like “ask for the departure date in an informal way and don’t make special use of politeness” and it should generate a question like “When do you want to leave?”. This process is also known as *Concept-to-Text* (or *Concept-to-Speech* if the result is being synthesised) and includes several steps and components, which we describe now.

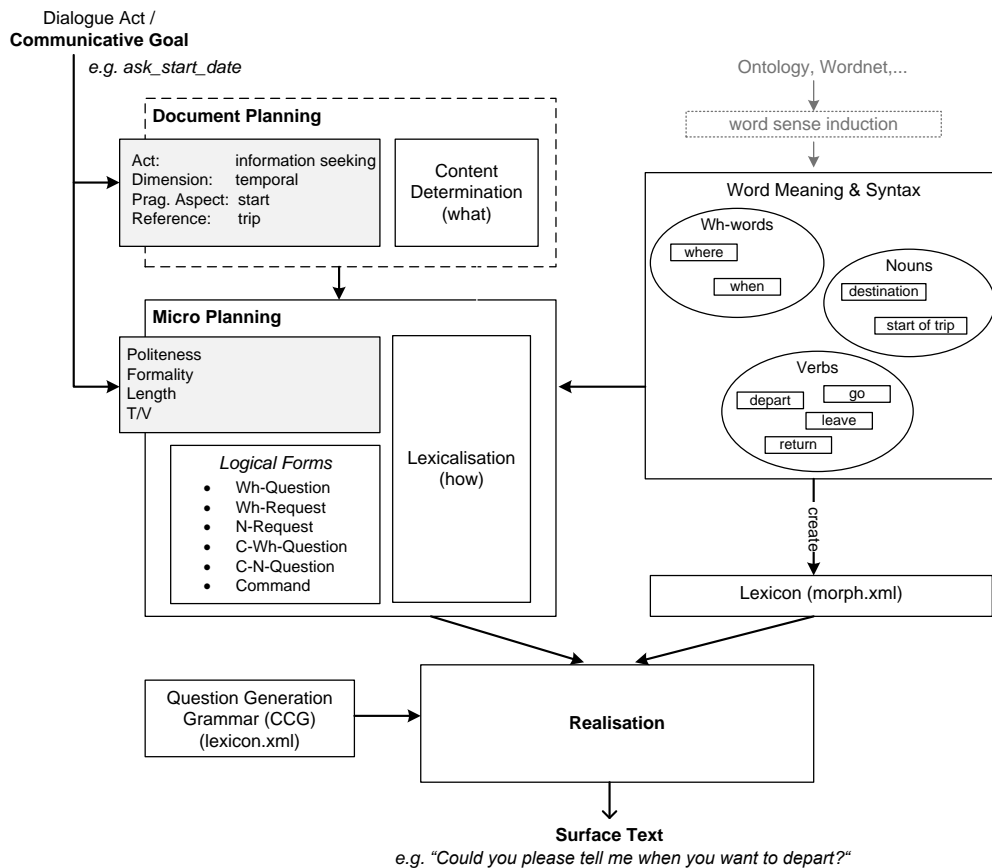


Figure 10.4: NLG system scheme

As you can see in Figure 10.4, we first need to define the communicative goal. This may be to ask for the start date and is usually provided by the dialogue manager. In a standard language generation process, the document planner determines how this goal can be delivered, i.e. what content needs to be selected (what should be said). In our case, this specification is given by the dialogue designer, i.e. specific dialogue acts are connected to a certain content. This requires the conceptual description of words and their meaning. Afterwards, the micro planner decides how this content can be expressed. This refers to the form (how the content should be formulated) and is known as lexicalisation (choice of words and grammatical style). In order to be able to choose a formulation that conveys a certain message, we need to define words, their meaning and parameters for their selection first (see Section 10.2.2). This also includes parameters for politeness and formality. Moreover, we need a set of semantic rules (logical forms) for the generation of different question types. With this information (lexical and semantic information) at hand we can create logical forms (comparable to Lexicalised Case Frames) that are used

as input for the realiser. The realiser then creates the surface text by applying grammar rules that define how to create text from logical forms.

Of course also referring expression generation and aggregation can be integrated into this approach. This is useful for the generation of complex questions. If the document plan consisted of the messages `tell_destination` and `ask_departure_date`, the aggregated result could be “*When do you want to go to London?*” instead of the separate sentences “*You want to go to London.*” and “*When do you want to depart?*”. By this means we can generate questions with implicit confirmations. We now take a closer look at the steps of the language generation process. Afterwards we describe our knowledge base and how we can use the results in dialogue systems.

10.2.1 Definition of a Generation Grammar

In order to generate different formulations, we have to define a grammar (used by the realiser), which we decided to specify in the CCG⁶ format. For a proof of concept we restrict ourselves to the following interrogative types:

- Yes-No-Question
- Wh-Question
- Wh-Request
- NP-Request
- Can-Wh-Question
- Can-NP-Question
- Command (NP)

As already mentioned, a categorial grammar does not consist of phrase structure rules that define how a wh-question or a wh-request can be created. Instead, we define a lexicon that describes how every word type can be combined, i.e. how categories can be aggregated to a more general category. A wh-word can be used to create an ordinary question (“*When do you want to go?*”) or can also be part of an indirect request (“*Can you tell me when you want to go?*”):

$$\begin{aligned} \text{when} \vdash \\ & \text{s[wh-question]/s[question]} \triangleright \text{question} \\ & \text{s[iwh-question]/s[b]} \triangleright \text{indirect request} \end{aligned}$$

In our example we apply the first category⁷. This is read as: *The word ‘when’ can become a sentence of type ‘wh-question’ if there comes a sentence of type ‘question’ from the right-hand side.* Since the definition of wh-words is not enough, we now take a look at how the right-hand category of the rule is defined. A question can be created with the help of an auxiliary verb like *do*:

⁶we use OpenCCG

⁷brackets indicate features that need to be unified

```
do ⊢
  s[question]/(s[b]\np)/np
  s[question]/s[dcl]
```

Again, we have different possibilities to use (or combine) the word *do*. We use the second rule (*the word ‘do’ can become a sentence of type ‘question’ if there is a declarative sentence on the right*). In order to create a declarative sentence, we need a verb. Generally, an intransitive verb is defined as $s\backslash np$. The feature **b** denotes a *bare infinitive* and **to** is an *infinitive with to* (Hockenmaier and Steedman, 2007, 2005). The combination of the lexemes *want*, *to*, and *go* leads to a category that becomes a declarative sentence if there is a **np** on the left. Figure 10.5 shows the complete application of the CCG categories.

In this example we have described the general methodology to parse wh-questions by

1	(lex)	when :- s[wh-question]/s[question]
2	(lex)	do :- s[question]/s[dcl]
3	(lex)	you :- np
4	(lex)	want :- (s[dcl]\np)/(s[to]\np)
5	(lex)	to :- s[to]\np/(s[b]\np)
6	(lex)	go :- s[b]\np
7	(>)	to go :- s[to]\np
8	(>)	want to go :- s[dcl]\np
9	(<)	you want to go :- s[dcl]
10	(>)	do you want to go :- s[question]
11	(>)	when do you want to go :- s[wh-question]

Figure 10.5: Simplified parse for ‘When do you want to go?’

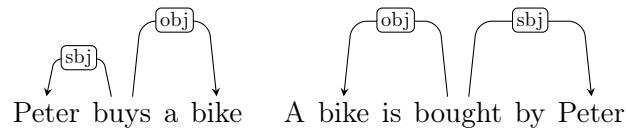
showing a very small subset of our lexicon. However, with the help of the grammar we do not want to parse sentences, but instead want to generate (or realise) them. The basis for the OpenCCG realisation process is logical forms, i.e. semantic representations of the sentence. Each syntactic category is associated with a logical form represented in a *hybrid logic dependency structure* that describes the relations between the words of a sentence; e.g. the transitive verb *buy* can be described as:

$$@e \text{ buy}, @e < \text{subj} > x, @e < \text{obj} > y$$

The verb *buy* has two relations, subject and object. The advantage of this semantic description is its independence from the syntactic form. The sentences “*Peter buys a bike*” and “*A bike is bought by Peter*” can be represented as:

$$\begin{aligned} @e(\text{buy} \wedge \\ < \text{subj} > (x \wedge \text{Peter}) \wedge \\ < \text{obj} > (y \wedge \text{bike})) \end{aligned}$$

or graphically as:



The logical form for the wh-question from our last example is depicted in Figure 10.6. We use thematic roles in order to specify the proposition of a question. In this case an *agent* wants a *theme*. We can also use semantic features to influence the style of the utterance, i.e. in this case we want to formulate an interrogative sentence with a second person singular agent. As already mentioned, logical forms are the basis for

```

1 s { stype=wh-question } :
2   @w0(when ^
3     <prop>(w3 ^ want ^
4       <mood>interrogative ^
5       <agent>(w2 ^ pron ^
6         <num>sg ^
7         <pers>2nd) ^
8       <theme>(w5 ^ go ^
9         <agent>x1)))

```

Figure 10.6: Logical form for ‘When do you want to go?’

the realisation process. To give you a better understanding for the following sections, we introduce another example in Figure 10.7. This logical form abstracts from grammar and

```

1 s { stype=c-question } :
2   @w0(can ^
3     <mood>subj ^
4     <actor>(w1 ^ pron ^
5       <num>sg ^
6       <pers>2nd) ^
7     <theme>(w2 ^ tell ^
8       <patient>(w3 ^ pron ^
9         <num>sg ^
10        <pers>1st) ^
11      <theme>(w4 ^ when ^
12        <prop>(w6 ^ want ^
13          <agent>(w5 ^ pron ^
14            <num>sg ^
15            <pers>2nd) ^
16          <theme>(w8 ^ leave ^
17            <agent>x1))))))

```

Figure 10.7: Logical form for ‘Could you please tell me when you want to leave?’

word position issues and just reflects the logical meaning of an utterance. Additionally, we need to know which words to use, i.e. OpenCCG requires a finished lexicalisation

process. While there are words that are only influenced by inflection (pronoun, sg, 2nd), we also have words that change the style of an utterance⁸. We already found that formality is strongly connected with the choice of words. That's why we need to generalise the semantic representation even further so that we can control the whole lexicalisation process.

10.2.2 Word Meaning Representation

The lexicalisation of questions involves different kinds of words (question words, nouns, verbs) depending on the question type that has been chosen. In order to describe question words, we can use the answer types from the AQD. Hereby, we connect the question word, e.g. *when*, to a certain class, in this case *temporal*, as illustrated in the following examples:

- *when* \rightsquigarrow `fact.temporal`
- *where* \rightsquigarrow `fact.named_entity.non-animated.location`
- *at what time* \rightsquigarrow `fact.temporal.time`

As you can see, underspecification enables us to use *when* in order to ask for times and dates. When asking "*at what time*" we refer of course only to a time and exclude dates. This approach works fine to describe simple interrogative adverbs. But for verbs we need more information than just the answer type in order to make an appropriate selection. A preliminary approach is the description of the word meaning by combining conjunctive descriptive features in a lexicon as indicated in the following examples:

- *go*: movement \wedge slow \wedge by feet
- *run*: movement \wedge fast \wedge by feet
- *drive*: movement \wedge by car
- *travel*: movement \wedge far away \wedge holiday

The basic idea is to describe words by other words, in this case by adjectives and nouns. The verb *go* can be described as a slow movement that is executed by feet. This approach leads us to the concept of *lexical decomposition* where we describe the meaning of a word by breaking it into more basic features (or *semes*). As illustrated in the following example, we can distinguish the meaning of *girl* and *woman* by the feature \pm adult.

$$\begin{aligned} \textit{girl} &= [+alive, +human, +female, -adult] \\ \textit{woman} &= [+alive, +human, +female, +adult] \end{aligned}$$

⁸indicated with bold face in Figure 10.7

This approach is mostly used with nouns and the description of verbs is more difficult. We may find words from the same word field (e.g. movement), but we do not find synonyms with the following set of features.

$$\begin{aligned} go &= [+movement, +slow, +feet, -car] \\ run &= [+movement, -slow, +feet, -car] \\ drive &= [+movement, +car] \\ travel &= [+movement, +holiday] \end{aligned}$$

More specifically, e.g. in the travel domain, a certain sense of the words *go* and *travel* can be synonymous, i.e. they can be used in the same context without changing the meaning (“*When do you want to go?*” vs. “*When do you want to travel?*”). Of course we can argue that these verbs are not strictly synonymous but rather constitute a hypernymous relation because *to travel* is more specific than *to go*. However, we can use both words without changing the general intention of the question. When we search *Wordnet* for synonyms of the verb “*go*”, we find two matching synsets⁹:

$$\begin{aligned} &\{\text{travel, go, move, locomote}\} \\ &\{\text{go, go away, depart}\} \end{aligned}$$

However, they do not perfectly fit our example, since *move*, *locomote* and *go away* would be bad choices for expressing the intention to ask for the travel dates. Although they all refer to a change of location, there are still pragmatic differences according to the domain and use case. So we need to find features that provide us with an abstract representation that enables us to select appropriate words for a given context.

A further approach is the distinction of verbs by *lexical aspects*¹⁰ as introduced by Vendler (1957). He differentiates *activity* (e.g. running, swimming, sleeping), *accomplishment* (e.g. running a mile, drawing a a circle, building a house), *achievement* (e.g. recognise, explode, win) and *state* terms (e.g. have, know, love), which could serve as additional semantic features. These aspects can be described by *durative* and *telic* characteristics. Verbs are *durative* if they describe an enduring action or state (e.g. I am sleeping/reading) or *punctual* if they are instantaneous, i.e. not progressive (if they describe the begin or end of an action, e.g. to find, to get ill, to arrive). Moreover, verbs can be *telic* or *atelic*, i.e. they have or do not have an endpoint. Sometimes, the behaviour of a verb to change a state (to awake vs. to sleep) is also considered an indicator. Vendler concludes that states and activities are durative and atelic, accomplishments are durative and telic, and achievements are punctual and telic (see Table 10.1).

The differentiation between activities and accomplishments is based on their telic char-

⁹“A *synonym set*; a set of words that are interchangeable in some context without changing the truth value of the preposition in which they are embedded.”, <http://wordnet.princeton.edu/man/wngloss.7WN.html>

¹⁰German: *Aktionsart* (translates as “*action type*”)

Table 10.1: Lexical aspects

	telic	durative	progressive (dynamic)
activity	-	+	+
accomplishment	+	+	+
achievement	+	-	-
state	-	+	-

acteristics. Activities “are processes going on in time [that] consist of successive phases following one another” (Vendler, 1957), e.g. *running*. We do not make any assumption “as to how long that running [...] will go on”. Accomplishments, on the other hand, have an endpoint and “proceed toward a terminus which is logically necessary to their being what they are” (Vendler, 1957). Hence, we exactly know when *running for a mile* will be finished. The differentiation between achievements and states is based on their durative behaviour: “achievements occur at a single moment, while states last for a period of time”. Hence, an achievement is also telic. The differentiation between an activity and a state can be told by the existence of continuous tenses (progressive). Although they are both durative, we can say “*I am running*” but not “*I am knowing*”. Consequently, the latter is a state. The difference between state and achievement can be recognised by checking if it is possible to ask “*How long...*”, i.e. states are durative, achievements are not, e.g. “*How long did you love her?*” vs. “*How long did you recognise her?*”.

Coming back to our example, we can see that the words *go* and *travel* are both durative and atelic. On the other hand, the verb “*depart*” would also be applicable¹¹ in this sentence although it is punctual, whereas “*move*” is not applicable although it is durative and atelic. Consequently, this differentiation alone is not of much help for finding synonyms (and telling apart different contexts).

$$\begin{aligned}
 go &= [+movement, +slow, +feet, -car, -telic, +durative] \\
 travel &= [+movement, +holiday, -telic, +durative] \\
 move &= [+movement, -telic, +durative] \\
 depart &= [+movement, +start, +telic, -durative]
 \end{aligned}$$

Let us shortly recap the descriptions we have at hand now. We began with the answer type, which we used to describe question words. We also have information about the general class (e.g. movement) and lexical aspects. But the actual meaning of the word still consists of rather randomly selected features like \pm slow and \pm feet. Furthermore,

¹¹we are aware that punctual verbs refer to a single date whereas durative verbs can ask for both the departure and the return date

feature vectors as used with the lexical decomposition are unfavourable if we combine aspects from different logical points of view (like type, temporal behaviour, meaning) because they are completely unstructured. We realise that we need more information about the verb, i.e. we need to find a common description for those words that we can use in the same (pragmatic) context. Thus, we propose to describe a word with its kind of usage, i.e. we describe in which situation a word can be used.

But let us first review relevant parts of the AQD. As you know, we can use the AQD to describe a question on a conceptual level. Hence, it is obvious that this description is also relevant to describe the words that we want to use to build that question. We have already determined that we need the answer type for the generation of questions because every question is formulated with regard to a certain dimension (e.g. time or location) of the answer. We have noted that one possibility to describe the meaning of a word can be done by specifying the context. When we take a look at the corresponding AQD level, we find the *role* of a question. By this means, we can describe that the answer type *date* may be used in the role *birthdate*. The role can be described by a *pragmatic aspect* and a *reference*. This description is closely connected to the word meaning, i.e. a question asking for a *date* in the role *start of life* refers to the noun birthday, as well as the word birthday refers to questions that ask for a date in the same role. Furthermore, every question is related to a domain. This is not part of the AQD but already included in the utterance description. Finally, we need syntactic information in order to ensure that the selected word matches the grammatical structure of the question type. For this purpose we use the *part of speech* (POS).

The resulting description makes essential use of the AQD levels *type* and *context*. It consists of the *part of speech* (needed to find syntactically matching words), a *domain*, a *dimension* (the AQD answer type), and the AQD *context* that consists of a *pragmatic aspect* that serves as a specification or constraint of the dimension and a *reference* that describes the relation towards the applicable situation.

- Part of Speech π
- Domain δ
- Dimension θ
- Context γ : Pragmatic Aspect $\alpha \wedge$ Reference ρ

The context includes outcomes from lexical aspects (durative and telic) that are simply called *begin* (for inchoative verbs) and *end* (for resultative verbs) in order to allow non-linguists (who will design the dialogue and the questions) to better understand the effect of this attribute. The combination of inchoative and resultative features indirectly marks a durative verb. Because the dimension is closely connected to the context and at the same time we can imagine different contexts for the same word (*go* can be used in the temporal dimension in the context of the start of a trip and in a local dimension in the context of the end of a trip), we combine these attributes as the *meaning* and specify them as a list. In short, every word (w type of π) $\in \delta$ is assigned a meaning μ , so that we

have the following features at hand to describe and select a syntactically matching word that can be used in the context specified by a given meaning:

- Part of Speech π
- Domain δ
- Meaning $\mu = \{(\theta_1, \alpha_1, \rho_1), \dots, (\theta_{n-1}, \alpha_{n-1}, \rho_{n-1}), (\theta_n, \alpha_n, \rho_n)\}$

In the following examples you can see how we can describe the meaning of words, together with some exemplary sentences. The definition of *go* reads as follows: It is a verb in the travel domain that can be used in a temporal context to describe the start of a trip (start date) or in a local context to describe the end of a trip (destination).

go (v) \in Travel

$\mu = \{(\text{fact.temporal.date, start, trip}),$
 $(\text{fact.namedEntity.nonAnimated.location, end, trip})\}$

- ▷ *Where do you want to go?*
- ▷ *When do you want to go?*

start (v) \in Travel

$\mu = \{(\text{fact.temporal.date, start, trip}),$
 $(\text{fact.namedEntity.nonAnimated.location, start, trip})\}$

- ▷ *Where do you want to start?*
- ▷ *When do you want to start?*

arrive (v) \in Travel

$\mu = \{(\text{fact.temporal.date, end, trip})\}$

- ▷ *When do you want to arrive?*

The word *accompany* is a verb in the travel domain that can be used when asking for the number of persons. Because we do not have a pragmatic aspect we just use a wildcard.

accompany (v) \in Travel

$\mu = \{(\text{fact.quantity, *, person})\}$

- ▷ *How many persons accompany you?*

Nouns follow the same scheme. The only difference is the part of speech.

departure city (n) \in Travel

$\mu = \{(\text{fact.namedEntity.nonAnimated.location, start, trip})\}$

- ▷ *Please tell me your departure city.*

When you take a look at question words, you can see that we have introduced a *general* domain and a wildcard referent because a question word is not specific to a certain domain and can refer to all sorts of situations.

when (whadv) \in General
 $\mu = \{(\text{fact.temporal}, *, *)\}$
▷ *When do you want to go?*
▷ *When do you want to have dinner?*

Apart from question words and related verbs or nouns (*when* do you want to *go*) we also have verbs like *want*, *tell*, *have* and *can* that cannot be related to a specific answer type. Here we can use the pragmatic aspect to denote the type of usage, e.g. a word that can be used in any dimension to indicate a *transfer of knowledge* with reference to any object.

tell (v) \in General
 $\mu = \{(*, \text{knowledge_transfer}, *)\}$
▷ *Can you tell me when you want to go?*

have (v) \in General
 $\mu = \{(*, \text{possession}, *)\}$
▷ *Do you have a customer card?*

The ontology extract in Figure 10.8 shows the basic description of the word *go*. You can see all the attributes from our specification. In order to be language independent and to introduce the possibility to create sentences with different levels of formality (we identified a connection between formality and the choice of words), we add the features *lex* and *formality*. Recall, that the *politeness* is influenced by subjunctive formulations and the addition of the word *please*, so this does not need to be reflected in the word meaning representation.

With this elementary definition of words we are now able to describe and generate simple questions. We basically describe a question by the meaning μ , i.e. the dimension (or answer type), a pragmatic aspect and a reference. A question that asks for the begin of a trip would be defined as follows:

$$\text{ask}(\text{fact.temporal.date}, \text{start}, \text{trip}) \tag{10.6}$$

According to our question style definitions from Section 10.2.1 and their related grammars, we choose either a verb or a noun to represent μ . Also neutral words like *want*

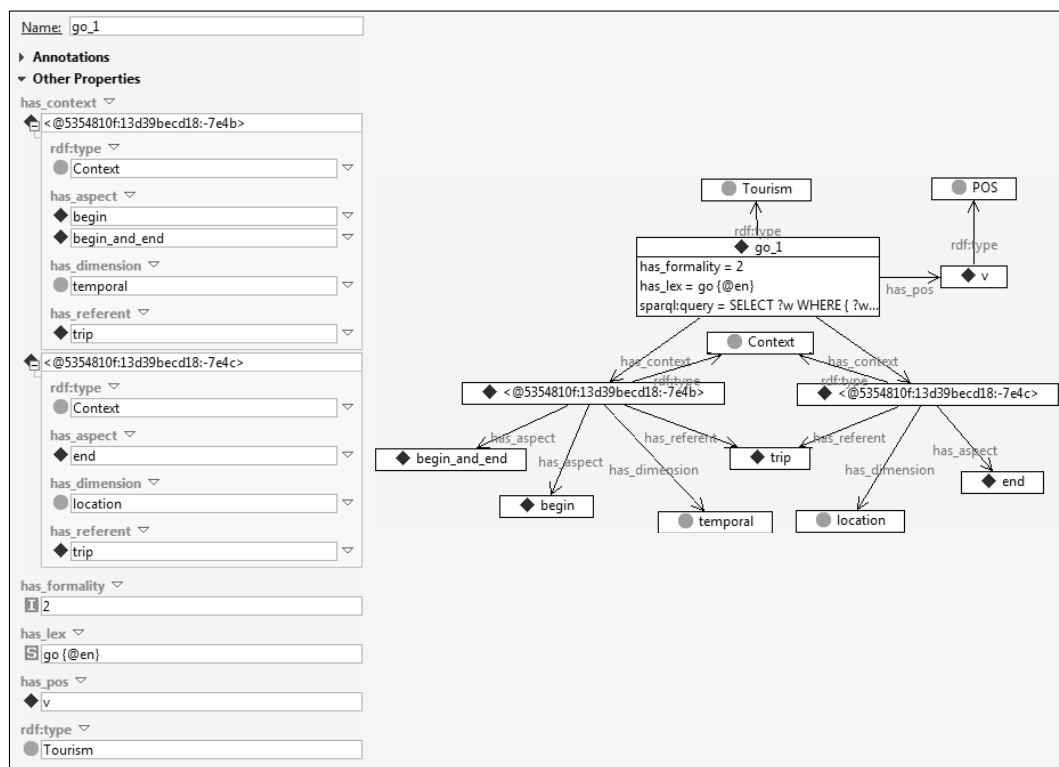


Figure 10.8: Extract from ontology

or *tell* are introduced in this step. In a very formal utterance *tell* could be replaced by *specify*. Apart from the formality, we also have to choose the correct question style according to the politeness. A high politeness value leads to the introduction of the word *please* and changes the mood from indicative to subjunctive. Moreover, we have a static list that assigns a politeness value to every question style. For example, a can-question is more polite than a request. The result of this step is the representation in Figure 10.7, which is – at the same time – the input for the OpenCCG realiser. In the next section we will describe the generation process in more detail.

10.3 Application in a Dialogue System

To illustrate the interfaces with the rest of the dialogue system we again refine our overview as shown in Figure 10.9. As well as the natural language understanding module, also the language generation module has a close connection to the dialogue manager, which selects content according to the current system state and discourse. Every communicative goal is a kind of SoDA that is further refined by the dialogue designer who

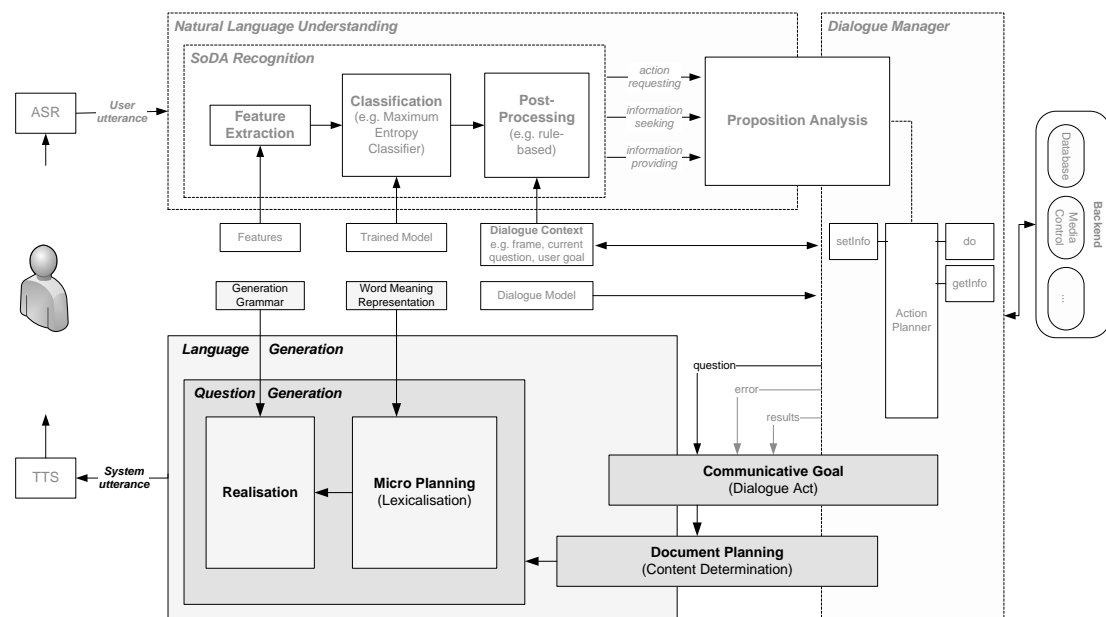


Figure 10.9: Question generation as part of the whole dialogue system

specifies an abstract description (an extract from the AQD) of the question to be asked. This description serves as document plan and is the input to the core of the question generator. Here, the system performs the actual lexicalisation and realisation.

We have already described how we can represent the word meaning. But how do we select an appropriate formulation that matches the politeness value? Let us first define the question *“Could you tell me when you want to leave?”* as an AQD¹²:

- Type:
 - Answer Type & Constraint: `fact.temporal.date{x>today}`
- Form
 - Question Type: `C-Question`
 - Language: `English`
 - Surface Modifier: `politeness=4, formality=2`
- Context
 - Purpose: `gather information`
 - Role
 - * Pragmatic Aspect: `start`
 - * Reference: `trip`

¹²for clarity only relevant parts of the AQD are shown here

We can now infer a *word meaning representation* from this AQD (which is basically just a subset), which is necessary to query the ontology. But how do we know which kind of words we need (depending on the question type we have a different syntax)? Here, the dialogue designer has two different possibilities: he can specify the question type directly (e.g. a C-Question) or he could just add surface modifiers (politeness and formality), as you can see in Figure 10.10. The system then infers which question type

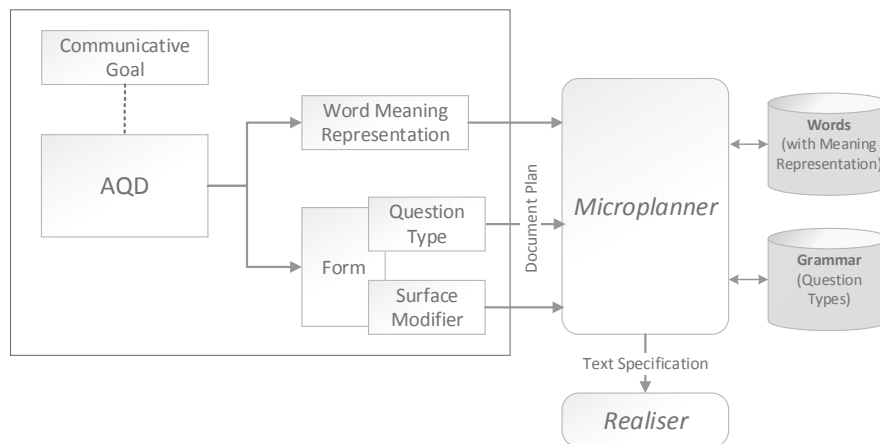


Figure 10.10: Question generation with reference to the AQD

matches those features best. For this purpose every question type is associated with a politeness value, i.e. a *Can-Wh-Question* has a higher politeness value than a *N-Request*. Depending on the chosen question type, the system first infers what type of word (POS) is needed. Afterwards, the ontology is queried so that we get a word carrying the indicated meaning. With this information we can construct a text-specification that is finally being realised.

In the next sections we introduce a programming interface that allows to easily define a question that will be realised according to the given parameters. We then evaluate the results in a short user study.

10.3.1 Programming Interface

As mentioned in the beginning, our aim is an easy integration of our approach in current dialogue systems. The programmer should not have to deal with complicated language generation issues. Instead, he should be able to use a simple interface. The management of the vocabulary is handled by the ontology. The following lines of code generate a dialogue with five system questions:

- ask for the start date of the trip
- ask for the end date of the trip
- ask for the departure city

- ask for the destination
- ask if the customer has a customer card

```
1 int intended_formality=2; // 1..5
2 int intended_politeness=1; // -2..5

4 questions.add(new Question("fact.temporal.date", "begin", "trip"));
5 questions.add(new Question("fact.temporal.date", "end", "trip"));
6 questions.add(new Question("fact.namedEntity.nonAnimated.location",
    "begin", "trip"));
7 questions.add(new Question("fact.namedEntity.nonAnimated.location",
    "end", "trip"));
8 questions.add(new Question("decision.yn", "possession", "
    customer_card"));
```

When setting the formality value to 2 and the politeness value to 1, we obtain the following results:

Dialogue 10.26:

S: *Please tell me your departure date!*
U: ...
S: *Now please tell me your return date!*
U: ...
S: *Please tell me your departure city!*
U: ...
S: *Now please tell me your destination!*
U: ...
S: *Do you have a customer card?*
U: ...

As you can see, every second utterance we insert a temporal connector (e.g. *now*) to make the dialogue appear more fluent. Moreover, the questions are formulated as requests, which means that we need to select nouns from the ontology. In the following example we have increased the politeness value to 4. You can see that the system now chooses C-Questions and makes use of verbs instead of nouns.

Dialogue 10.27:

S: *Can you please tell me when you want to go?*
U: ...
S: *Can you now please tell me when you want to return?*
U: ...
S: *Can you please tell me where you want to start from?*
U: ...
S: *Can you now please tell me where you want to go?*
U: ...
S: *Do you have a customer card?*
U: ...

When also increasing the formality to 4, we observe a different choice of words, for example in the first utterance:

Dialogue 10.28:

S: *Can you please tell me when you want to depart?*

U: ...

We can see that, due to the static relationship between politeness and question style, almost every utterance has the same formulation within each dialogue. That's why we introduce a *politeness* variation that automatically varies the politeness around a given value. When you compare the following Dialogue 10.29 with the original Dialogue 10.26, you can see the variation in the formulation which follows the pattern:

$$p'_i = \begin{cases} p - 1, & \text{if } n \bmod 3 = 0 \\ p, & \text{if } n \bmod 3 = 1 \\ p + 1, & \text{if } n \bmod 3 = 2 \end{cases}$$

e.g. for $p = 2$: $p'_1 = 2, p'_2 = 3, p'_3 = 1, p'_4 = 2, \dots$

Dialogue 10.29:

S_1 : *Please tell me your departure date! ($p'=2$)*

U: ...

S_2 : *And when do you want to return? ($p'=3$)*

U: ...

S_3 : *Departure city please! ($p'=1$)*

U: ...

S_4 : *Now please tell me your destination! ($p'=2$)*

U: ...

S_5 : *Do you have a customer card? ($p'=3$)*

U: ...

In this case we make use of a mix of requests, commands and questions. The next example is an informal and medium politeness dialogue. According to the politeness variation algorithm, the second system utterance should be slightly more polite than the first one.

Dialogue 10.30:

S: *Can you tell me when you want to depart?*

U: ...

S: *Can you now please tell me when you want to return?*

U: ...

S: *Where do you want to depart?*

U: ...

S: *Can you now tell me where you want to go?*

U: ...

S: *Do you have a customer card?*
 U: ...

However, it still makes use of the same formulation (C-Wh-Question). Instead of varying the question type, we have added the word *please*. For an overview which formulation corresponds to which politeness level, you may refer to Section A.2 on page 205.

10.3.2 Evaluation of Generated Example Interrogatives

After having demonstrated the functionality of the proposed system, we now evaluate its plausibility and effects. We have asked 26 human judges to evaluate the dialogues' naturalness and politeness. During the test we presented the participants four dialogues (see Figure 10.11) with different politeness levels. The users had to sort the dialogues according to their politeness. Afterwards, they had to indicate which of the dialogues might have been uttered by a human and which dialogue they prefer.

A	B	C	D
A: When do you want to set off?	A: Can you please tell me when you want to set off?	A: Departure date?	A: Departure date please!
B: ...	B: ...	B: ...	B: ...
A: Can you now tell me when you want to return?	A: Could you now please tell me when you want to return?	A: And the return date?	A: Now please tell me your return date!
B: ...	B: ...	B: ...	B: ...
A: Please tell me your departure city!	A: Can you tell me where you want to start from?	A: Departure city?	A: Tell me your departure city!
B: ...	B: ...	B: ...	B: ...
A: And where do you want to go?	A: Can you tell me where you want to start from?	A: And the destination?	A: And the destination please!
B: ...	B: ...	B: ...	B: ...
A: Do you have a customer card?	A: Can you now please tell me where you want to go?	A: Customer card?	A: Do you have a customer card?
B: ...	B: ...	B: ...	B: ...
	A: Do you have a customer card?		
	B: ...		

Figure 10.11: Survey

We began with the sorting task. In general the participants correctly classified the dialogues according to our intended politeness levels, and 46% put the dialogues completely in the right order (*C, D, A, B*). The participants classified the two more impolite dialogues as more polite than they are, and the two more polite ones as less polite, as shown in Table 10.2, but this can possibly be explained by people's tendency to choose values in the middle of a scale rather than at either extreme.

Table 10.2: Dialogues sorted by politeness scores

<i>dialogue</i>	<i>original scores</i>	<i>mean user scores</i>	Δ	<i>correctly classified by</i>
C	1.0	1.8	+0.8	62%
D	2.0	2.2	+0.2	62%
A	3.0	2.7	-0.3	58%
B	4.0	3.3	-0.7	65%

Now we asked the participants which dialogue they like most. 77% of them preferred dialogue A, 19% preferred dialogue B and 4% dialogue D. When we quantify the preferred dialogues with the corresponding politeness scores and normalise the result (see Equation 10.7), we get an average preferred politeness score of 3.2 (original score) respectively 2.8 (user score), which again refers to dialogue A.

$$\frac{1}{|user|} \sum_{i=1}^{|dialogues|} score_i \times |votes_dialogue_i| \quad (10.7)$$

In the last step we asked the users to indicate which dialogue might have been uttered by a human. It was also possible to say none or to give multiple answers. 88% of the participants think that dialogue A could have been uttered by a human, 42% think dialogue B might be of human origin, 19% declare dialogue C and 4% dialogue D as human. This evaluation confirms that the system is able to create questions in different politeness levels and that these levels are correctly identified by the users. As already mentioned, 88% of all participants thought that dialogue A was a dialogue uttered by a human although it was automatically generated by our system. However, at least every fifth user likes dialogue B most, which supports our assumption that adaptive language generation is necessary to increase user satisfaction.

10.4 Summary

In this chapter we have proposed a model to create system utterances in different politeness and formality levels. This model serves as our basis for improving dialogue systems and dialogue development environments with respect to adaptive and human-like formulations of system utterances with little work for the developer. Based on a categorial grammar we have created seven different interrogative utterance types that represent the syntactic forms for different politeness values. An ontology defines the meanings and formality levels of the used words.

To verify the operability of our model, we have developed a programming interface that is used to define concepts which are then realised according to the politeness and formality parameters. A final evaluation of the generated utterances proves that our model is a valid methodology to support concept-to-text with respect to the generation of system questions.

A goal without a plan is just a wish.

(Antoine de Saint-Exupéry, 1900 – 1944)

11

Building a Dialogue Model

In the last chapters we have introduced some techniques – like dialogue acts, answer types and language generation – to improve the creation of natural dialogues. We now incorporate them into a dialogue model that can be automatically processed by a dialogue manager, which serves as the foundation for our goal to facilitate the design of natural dialogues.

11.1 Dialogues and Tasks

Dialogues may be short and precise in that they cover only a single topic, e.g. travel information, as the following example illustrates:

Dialogue 11.31:

S: *Hello, how may I help you?*

U: *I'd like to book a trip.*

S: *Where do you want to start?*

U: *In Hamburg.*

S: *And where do you want to go?*

U: *Edinburgh.*

S: *Can you now tell me when you want to leave?*

U: ...

However, this kind of dialogue occurs rather seldom in human interaction. Subdialogues, which introduce further topics, need to be modelled and understood as well. The next exemplary dialogue contains three topics: booking information, city information and weather information.

Dialogue 11.32:

S: *Hello, how may I help you?*

U: *I'd like to book a trip.*

S: *Where do you want to start?*

U: *In Hamburg.*

S: *And where do you want to go?*

U: *Um, can you tell me something about Paris?*
S: *Paris is the capital of France and is called the city of love. It offers a variety of interesting sights, e.g. the Eiffel Tower and the Louvre where you can see the Mona Lisa.*
S: *Please tell me where you want to go!*
U: *Well, how is the weather in Paris?*
S: *22°C and sunny.*
S: *Where do you want to go?*
U: *OK, Paris then.*
S: *Can you now tell me when you want to leave?*
U: ...

As you can see, the main task is to get information about a trip. While the user initiates subdialogues by ignoring the system's question and asking for different information instead, the system keeps track of the original goal and comes back to the last question that has not been answered yet.

These topics all have different goals and contain different questions that need to be answered before these goals can be fulfilled. We call every topic with a certain goal a *task*. Every task can be a subdialogue in another task. Through the use of mixed initiative we can also achieve subtasks where the dialogue system does not need to pose any question because the user has given all the required information in his initial request (as with the weather task in Dialogue 11.32).

Every *task* can be described by:

- a name / identifier,
- a domain,
- a selector,
- a dialogue act that describes the type of the task,
- a list of questions (ITO),
- an action to be executed when all required information has been gathered.

The task about the booking information may be called *getTripInfo* and is associated with the *tourism* domain. The *task selector* helps to detect if a current task is responsible for the user utterance and activates it. Moreover, a system-oriented dialogue act describes the intention of the task and the type of the action. In this case, it's an information-seeking dialogue act because the user wants to get information about a trip and thus the system needs to retrieve information from the back-end. Every task consists of pieces of information that need to be asked for, e.g. departure city, travel dates or the number of persons. Only if these questions have been answered, the system is able to formulate a back-end request (i.e. the system executes the specified action).

Every task is part of a dialogue. On the dialogue level general information that apply for the whole dialogue can be specified. This includes:

- name
- start task
- language
- global politeness
- global formality
- dialogue strategy
- usage of dialogue acts

The *name* is an identifier and can be used to load the correct dialogue description. By defining a *start task* we specify the first question to be asked. If not defined, the system will automatically choose the first question from the first task. *Global politeness*, *global formality* and *language* are important for the language generation component and define the style of the dialogue if not overwritten locally (on the ITO level; see Section 11.2 on page 155).

Moreover, the *dialogue strategy* is described on the dialogue level. Instead of choosing from *system initiative*, *user initiative* or *mixed initiative*, the dialogue designer can set the preferences on a more granular level. This is due to the fact that the term *mixed initiative* is not unambiguously described in literature. Sometimes, mixed initiative refers just to the ability to choose the order in which the questions are answered (often any frame-based system). In other cases mixed initiative also includes corrections, over-informativeness or the ability to switch tasks. So instead of choosing a generic class, the following precise features can be selected in this model.

- switch tasks
- over-informative answers
- different question
- correction

The switching of tasks refers to the ability to use subdialogues, e.g. to ask for the weather while being in the trip information task. Over-answering is the ability to give more information than has been asked for. In this case, at least the current question needs to be answered. However, if *different question* is set to *true*, it is also possible to completely ignore the current question and provide the system only with information concerning a different question. The *correction* flag is used to allow the change of information that has already been given by the user. Based on these settings we can achieve different effects which may be useful for lecture or to enforce a certain behaviour of the dialogue system. Consider the following example.

Dialogue 11.33:

- S: *Where do you want to depart?*
U: *I'd like to depart in Berlin.*
S: *And where do you want to go?*
U: *I'd rather want to depart in Hamburg.*

When corrections are switched off, the result will be a trip from Berlin to Hamburg which is obviously wrong. If we allow corrections, we need to recognise the answer as a correction instead of an answer to the question for the destination. Consequently, the activation of corrections may require a more sophisticated NLU component, as the following dialogue shows as well:

Dialogue 11.34:

- S: *Where do you want to depart?*
U: *I'd like to depart in Hamburg.*
S: *And where do you want to go?*
U: *To Paris.*
S: *When do you want to depart?*
U: *I'd rather like to go to Vienna.*

If the NLU component is only realised as a mere keyword spotter (i.e. city names) and corrections are enabled, the system may regard the utterance *Vienna* as a correction for the first question because without the context given by the question, the system may not be able to identify the answer as information about the destination city and thus applies it to the first question that asks for a city.

The setting *different question* enables the user to ignore a question and give an answer to a different question instead.

Dialogue 11.35:

- S: *Where do you want to depart?*
U: *I'd like to pay less than 700 €.*

This should not be confused with over-informativeness, where the user gives more information than required and does not ignore the question.

Dialogue 11.36:

- S: *Where do you want to depart?*
U: *I'd like to depart in Hamburg next Monday.*

The switching of tasks enables subdialogues and creates a more natural dialogue. In today's systems we often encounter dialogues like this:

Dialogue 11.37:

- S: *When do you want to depart?*
U: *How will the weather be in Vienna next Sunday?*
S: *I did not understand that. When do you want to depart?*

This is frustrating for the user and prevents him from achieving his goal. The following dialogue recognises the user's intention and helps him to accomplish what he asks for.

Dialogue 11.38:

- S: *When do you want to depart?*
U: *How will the weather be in Vienna next Sunday?*
S: *It will be 26°C and sunny.*
S: *When do you want to depart?*

In this connection, also dialogue acts become important. Without activated dialogue acts and only a keyword spotter as NLU component, the system will identify the user's utterance *Vienna* as an answer to the question for the destination.

Dialogue 11.39:

- S: *Where do you want to go?*
U: *How will the weather be in Vienna next Sunday?*

However, with the use of SoDA (see Section 8.4 on page 100) we can identify the utterance as an information-seeking request and thus infer that it does not qualify as an answer to the current question. Another example refers to the differentiation between information-seeking and action-requesting utterances. While the intention of the first dialogue is to get the state of the lamp, in the second dialogue the user actually wants to switch it. However, without SoDA also the first dialogue would result in a state change, which is not what the user wanted.

Dialogue 11.40:

- U: *I'd like to know if the lamp is switched on.*

Dialogue 11.41:

- U: *Please switch the lamp on!*

The correct interpretation of these utterances could also have been realised with a more sophisticated NLU component. However, with the use of SODA we are able to build simple systems that rely on keyword-based approaches.

As you can see, with the help of some global dialogue parameters we can dramatically influence the dialogue behaviour and simulate different types of systems without changing the actual questions. In this section we have described the dialogue- and task aspects of the model. In the next section we focus on *Information Transfer Objects*.

11.2 Modelling of Information Transfer Objects

We have learned how to specify the meaning of questions with the help of the AQD. We have also seen how we can use this specification to generate system questions. Moreover, the *type* dimension of the AQD allows us to describe possible answers. Because of the strong relationship between a question and its answer, we propose to model both within

the same object, which we call Information Transfer Object (ITO). An ITO can be described by:

- its **name**, i.e. a unique identifier, e.g. `getDepartureDate`
- a **group** that describes the proximity of two ITOs (departure date and return date are semantically closer than departure date and number of persons)
- an **index** that determines a certain order
- a flag that defines if the ITO is **required** or optional
- a flag that defines if the system should **generate the question** or if a static, user-defined default question should be used (fallback)
- an **AQD** that describes the question and its possible answer types (this information is necessary for language generation and parsing)
- a **fallback question** that is used if the question can not be generated from the AQD or if the dialogue designer decides not to use the language generation component

The following example demonstrates how an ITO can be specified. In this case, we want to model the departure city.

```
[ name :      getDepartureCity
  group :      1
  index :      0
  fallback :   Where do you want to start?
  useLG :      yes
  AQD type :   fact.named_entity.non_animated.location.city
  AQD context : begin,trip
  AQD form :   n/a ]
```

Figure 11.1: Information Transfer Object

Remember, that ITOs are used to describe two aspects of an information unit: asking for information and processing the answer. We first name the ITO `getDepartureCity`, assign it to a group, e.g. `1`, and provide an index, e.g. `0`, which makes it the first question to be asked in this dialogue. Furthermore, we create a fallback question (“*Where do you want to start?*”) that will be used if the language generation fails or if the flag is set to false. With the help of another flag, we can specify whether this ITO is optional and is only used to ask for further, but not necessary, information (e.g. in case of too many results or ambiguity). The automatic question generation module, as well as the natural language processing component, need information about the type of the ITO. That’s why we assign an AQD to each ITO. Recall, that an AQD consists of three dimensions: type, context and form. The question for the departure city is of the answer type

`fact.named_entity.non_animated.location.city`. This allows the natural language understanding component to select the correct parser (in this case *cities*). However, in order to generate a question, we need more information about the context, i.e. in what way we speak about a city. In the example at hand we refer to the begin of a trip. The optional *form* of the AQD can be specified locally and overrides any value that might be set on the dialogue level.

11.3 Actions and Follow-Up Questions

When all necessary information has been gathered from the user, the system will eventually invoke some kind of back-end action. This may be querying a database, updating information in a file, calling a web service, getting information from the web or switching a device. Depending on the type of action, we need different attributes that will be introduced in Chapter 12. However, every action has a *result mapping* that defines system answers based on the value of the result. This allows us to return appropriate utterances, e.g. “*This number was too small*” or “*This number was too big*” in the case of a number guessing game.

Follow-Up Questions are asked after an action has been executed and are used to get to know what the user wants to do next. After booking a trip, the system may ask whether the user wants to book another trip or whether he wants to have information about possible activities at the destination. In the number guessing game example a follow-up question might be “*Do you want to play again?*”. Follow-up questions contain an ITO that specifies the actual question and its possible answers as well as an *answerMapping* that defines which user-answer leads to which subsequent task.

11.4 Resulting Dialogue Specification

We now have a list of elements with corresponding attributes at hand and need to formalise the model for being able to process it automatically. For a system-independent definition we decided to use an XML notation. The following extract shows the definition of a task to get information about the weather. A *bag of words task selector* is used to activate this task if the user mentions the words *weather*, *forecast* or *temperature*. Apart from the bag of words selector, also more sophisticated NLU approaches could be used, but have not been modelled yet in this system.

```
<task name="getWeatherInformation">
  <selector>
    <bagOfWordsTaskSelector>
      <word>weather</word>
      <word>forecast</word>
      <word>temperature</word>
    </bagOfWordsTaskSelector>
  </selector>
  <itos>
    <ito name="getWeatherCity">
      <AQD>
        <type>
```

```

<answerType>fact.named_entity.non_animated.location.city</answerType>
</type>
</AQD>
<fallback_question>For which city do you want to know the weather?</
  fallback_question>
<group>0</group>
<index>0</index>
<required>>false</required>
<useLG>>false</useLG>
</ito>
</itos>
<action>
  <httpAction>
    <returnAnswer>>true</returnAnswer>
    <utteranceTemplate>The temperature in %getWeatherCity is #result degrees.</
      utteranceTemplate>
    <method>get</method>
    <params>q=%getWeatherCity&mode=xml&units=metric</params>
    <url>http://api.openweathermap.org/data/2.5/weather</url>
    <xpath>/current/temperature/@value</xpath>
  </httpAction>
</action>
</task>

```

The task at hand consists of only one ITO — the city that the user wants to know the weather of. In this case, language generation is not used and instead a pre-rendered question will be asked when the task gets activated. Furthermore, it is also possible to complete the whole task without answering a single question. This behaviour occurs if the user combines the selection of the task with further information like *“How will the weather be in Vienna?”*. Through the specification of the answer type within the AQD, the system is able to select the correct NLU module. When all information is provided (i.e. the city name), the dialogue system invokes the specified action. In this task, the system will send a **GET** request to the *openweathermap.org* service and fills the utterance template with the result. Hence, a possible answer may be *“The temperature in Sofia is 30°C”*.

Of course, this task is embedded into a dialogue. For a full example, please refer to Section A.3 in the Appendix. For reasons of clarity, we apply an XSL transformation to visualise the model, as can be seen in Figure 11.2. The structure of the XML file and thus an overview of the model can be best described by an XML schema (XSD), as you can partly see in Figure 11.3.

The dialogue element is the root of every document. It contains one or several tasks. Every task needs to have a selector in order to determine which task is responsible for the user utterance and thus reduce ambiguity. The elements *ITO* and *Action* are optional. This may seem strange on a first look, but introduces a great flexibility as the following examples demonstrate.

No action: A task that does not aim at executing a certain (back-end) action is needed to realise welcome messages or menus (choice questions). Consequently, a task without actions has no influence on the external world and instead only changes the internal context.

No ITO: A task that does not contain any questions or answers is possible when the naming of the task itself refers to an action without needing to ask for further information. The task *getLightbulb*, for example, directly leads to the retrieval of the current state of the light bulb, without the need to provide any further details.

```

Dialog: dummy2
start_task_name: start global_language: en global_politeness: 2 global_formality: 2 useSODA: true allowSwitchTasks: true allowOverAnswering: true

Task: start
Selector > bagOfWordsTaskSelector
hello

ITO: welcome
answerType: open_ended
fallback_question: How may I help you? group: 0 index: 0 required: false useLG: false

Task: getTripInformation
Selector > bagOfWordsTaskSelector
travel book journey trip

ITO: getDepartureCity
reference: trip specification: begin temporalOpener: false answerType: fact.named_entity.non_animated.location.city
fallback_question: Where do you want to start? group: 0 index: 0 required: false useLG: true

ITO: getDestinationCity
reference: trip specification: end temporalOpener: false answerType: fact.named_entity.non_animated.location.city
fallback_question: Where do you want to go? group: 0 index: 0 required: false useLG: true

ITO: getNumberOfPersons
answerType: fact.quantity
fallback_question: For how many persons? group: 0 index: 0 required: false useLG: false

ITO: getDate
reference: trip specification: begin temporalOpener: false answerType: fact.temporal.date
fallback_question: When do you want to leave? group: 0 index: 0 required: false useLG: true

ITO: getDate
reference: trip specification: end temporalOpener: false answerType: fact.temporal.date
fallback_question: When do you want to come back? group: 0 index: 0 required: false useLG: true

Action > groovyAction
  • resultMappings:
    ◦ message: #price Euro for %getDestinationCity is cheap! resultValue: 257 resultVarName: price
    ◦ message: #price Euro for %getDestinationCity is expensive! resultValue: 1000 resultVarName: price
  • returnAnswer: true
  • utteranceTemplate: This trip from %getDepartureCity to %getDestinationCity costs #price Euros.
  • code: executionResults.put("price", "257")

Task: getWeatherInformation
Selector > bagOfWordsTaskSelector
weather forecast temperature
    
```

Figure 11.2: Extract of a dialogue example

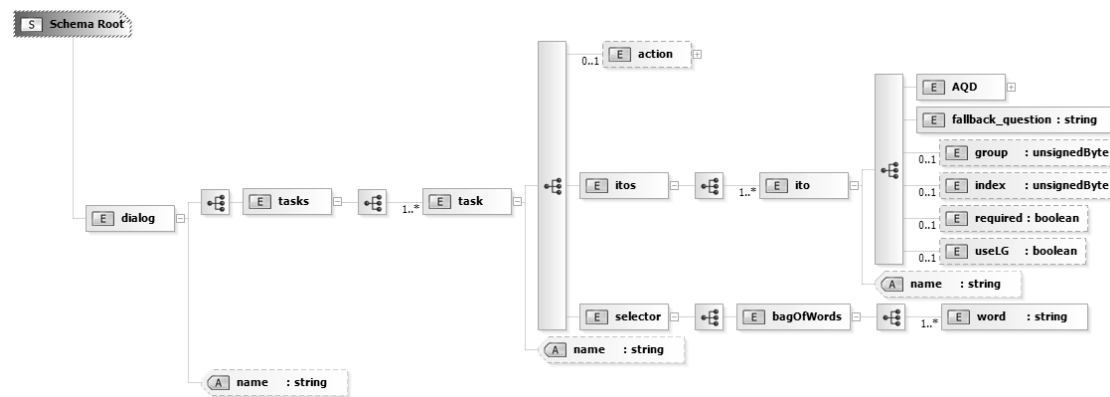


Figure 11.3: Extract of the dialogue model as schema definition

11.5 Adaptivity

In Section 3.2.4 we have identified characteristics of natural dialogue systems and found that adaptation is important to contribute to a pleasant and effective interaction between user and system. This has been confirmed by our user studies (see Part II). Our results show that users have different preferences and that these also change over time. Sometimes the preferred behaviour is also dependent on the task: the control of a lamp in a Smart Room is more likely to result in command-oriented language than a consultation in a travel agency scenario. Also, the detection of difficulties should be reflected by a change of the dialogue system’s behaviour. Hence, the dialogue model should provide the possibility to adapt the dialogue within the scope of a given set of features, e.g. :

Initiative / Dialogue Strategy: The system could start with an open-ended question in a mixed-initiative style and could change to a system-directed style in case of errors.

Confirmation: If the recognition confidence falls below a certain threshold, the system could change from implicit to explicit confirmation.

Speech synthesis / Voice: Gender, age and speed could be selected by a user model.

Language: Also the language can be switched according to the user’s language.

Formulation: Different styles for different users can improve satisfaction and make the system appear more human, which reduces entry barriers for new users, elder or technophobe people. This can be realised by adapting politeness and formality through the choice of words, sentence length and grammatical style.

Moreover, the formulation style of the system has influence on the style of the user answers. Bell (2003) argues that we can *“influence users to behave in a certain way,*

for instance by implicitly encouraging a speaking style that improves speech recognition performance”. Jokinen and Wilcock (2007) suggest an inclusion of context information in the case of low confidence levels. This refers to different confirmation styles depending on the recognition confidence. In their example, the answer to the question “When will the next bus leave for Miami?” could be “2.20pm”, “It will leave at 2.20pm” or “The next bus to Miami leaves at 2.20pm”. As we have pointed out, every user is different. In order to support the user at best, the user interface needs to adapt to the user. Otherwise, we can only find a common denominator, but not an optimal interface style. Every dialogue could start in a formal and mixed-initiative way and adapt to the user’s style of speaking over runtime. The realisation of this strategy has to be done by the dialogue engine. The dialogue model itself can only provide parameters to describe the intended behaviour of the dialogue. This includes supported languages, type of initiative and formulation/politeness.

11.6 User Model

An adaptive dialogue system adapts to the user’s style of interaction. Thus we need to model the user and analyse its current behaviour, that can then be used to infer a suitable dialogue strategy. In accordance with the model, we can set parameters for:

- initiative
- formulation (politeness/formality)
- language

Parameters for the preferred voice can be added easily. However, a confirmation style parameter is missing at the moment¹. The mentioned parameters can be set in advance by the dialogue designer to realise a certain behaviour of the dialogue system. Another approach is the automatic adaptation of these values depending on the user’s input. This does not require any change in the model but calls for a respectively sophisticated dialogue engine that analyses the user’s behaviour and reacts correspondingly.

We have already learned that the term *natural* not necessarily refers to the usage of the most complex technology and dialogue strategy but can also be described by a system behaviour that matches the user’s expectations. Even if the expectations are low and the user only wishes for a simple command-and-control system. The following list provides some strategies that may result in a (more) natural dialogue system:

- the fewer words are used in the request, the more formal and short should be the response
- the use of *please* or conjunctive forms should result in a higher politeness

¹Right now we can call the `ask`-method on any ITO. In the future, a `verify`-method will be added, which results in a question like “Do you really want to go to London from 1 until 14 August?”.

- the system should make use of a formality grade (especially T-V distinction) that reflects the user's choice
- the system should use the language of the user if available
- the system should change to system initiative if it encounters difficulties in interpreting the user utterance

However, the implementation of an adaptive dialogue engine, that analyses and reacts on the style of the user utterances is not the focus of this thesis and is left for future work.

11.7 Summary

In this chapter, we have combined the results from the preceding chapters and built a dialogue model that allows the dialogue designer to specify a dialogue that makes use of dialogue acts, answer types and language generation. Furthermore, the dialogue has been structured into different *tasks*, each of which contains three main parts: a *selector* to activate the task, several *ITOs* to specify the information demand, and an *action* that executes the desired command. We have defined all the necessary attributes and created an XML-based specification of the proposed model. This serves as input for the dialogue engine which will be addressed in the following chapter. Several parameters define the dialogue behaviour (type of initiative) and are interpreted by the dialogue engine.

An idea that is developed and put into action is more important than an idea that exists only as an idea.

(Buddha)

12

Dialogue System and Development Environment

In the last chapters, different topics on how to improve current dialogue systems have been addressed. This includes a simpler approach to dialogue acts and their relation to the back-end, the definition of a linguistic type hierarchy to support predefined parsers, the conception and implementation of a question generation component, the definition of a dialogue model that can be automatically processed by a dialogue manager, as well as two user studies to identify the problems of current dialogue systems. As Bringert (2008) points out, there are three major problems with current dialogue systems:

1. They are not natural enough.
2. They are not usable enough.
3. They are not cheap enough.

We believe that these problems depend on each other. Current dialogue systems are not usable enough because they are not natural enough because the development of natural dialogue systems is too expensive. We now want to incorporate our findings into a development environment that addresses these problems. We suppose the following features to contribute to the creation of an easy-to-use development environment for natural dialogues.

- separate the dialogue manager/system from the dialogue specification
- focus on the definition of the information units that are required to fulfil a task by combining the description of questions and their possible answers
- choose from a set of dialogue strategies instead of defining them repeatedly
- use language generation methods in order to abstract from language and style and make the creation of adaptive systems possible
- use predefined natural language understanding modules that can be selected by an abstract question description

In the context of this thesis, we focus on information-seeking dialogues, i.e. dialogues that collect a certain amount of information before an action is executed and reacted upon.

We have already defined the elements that a dialogue consists of and by this means proposed a model of how to specify a dialogue. This dialogue description now serves as the input for the dialogue system, which we are going to describe now.

The natural dialogue system (NADIA) consists of different modules which we will look at separately. Apart from the core of the system — the *dialogue engine* that runs the dialogue — the system provides different user interfaces and a dialogue development environment¹. As you can see in Figure 12.1, the engine provides two integrated access possibilities: a local console and a REST service. The REST interface can be directly accessed by simple POST requests. However, this option is only recommended for developers. Usually this layer is hidden by the user interfaces. Consequently, the end user will interact with the system by using a web page or an instant messenger like Skype. When the dialogue engine is started, it can run a default dialogue or it can be loaded

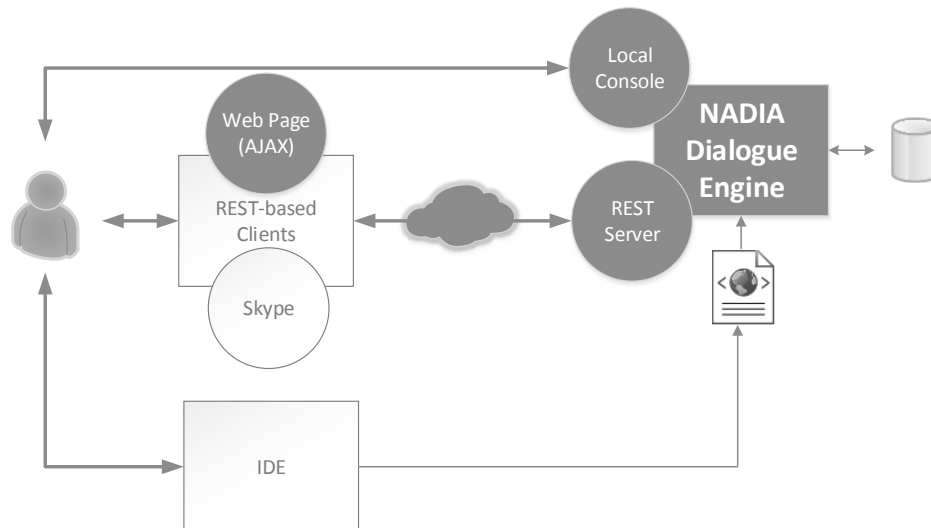


Figure 12.1: Schematic overview of the Natural Dialogue System

with a custom dialogue definition that conforms to the XSD that we defined in the last chapter. This definition can be created with the help of a Java library, as a plain XML file or by using the dialogue development environment.

¹the development of the DDE is dealt with in the thesis by Sötebier (2013)

12.1 User Interfaces and Communication with the Dialogue Engine

In order to access the dialogue system, we provide a web-based client that you can see in Figure 12.2. Through the use of a REST interface, developers can also create own clients, e.g. a Skype Client that has been created by Thielbeer (2013).

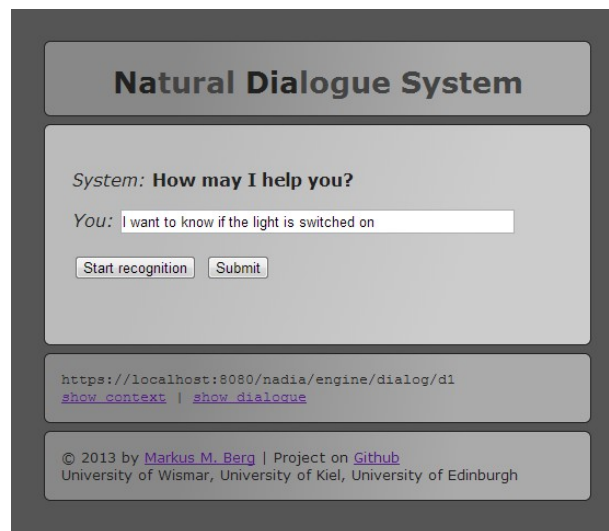


Figure 12.2: Web Client

Every client first needs to initialise a new dialogue. This is done by sending a POST request to the dialogue engine. The system responds with the first question and a new dialogue instance (i.e. a new URL). This communication process can be best explained by means of the sequence diagram in Figure 12.3. A call to `http://localhost:8080/nadia/`

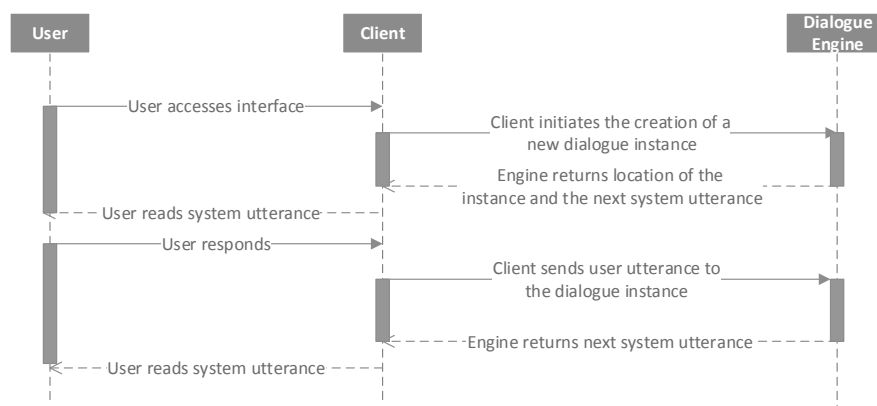


Figure 12.3: Client-Server-Communication

`engine/dialog` creates a new dialogue instance (HTTP Status Code 201, CREATED). In accordance to RFC 2616, the response contains a new location header (see Figure 12.4), which is used by the user agent in any subsequent interaction.

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: http://localhost:8080/nadia/engine/dialog/d1
Content-Type: text/plain
Transfer-Encoding: chunked

How may I help you?
```

Figure 12.4: Response log: initialisation of a new dialogue instance

So the user agent needs to pass the data (i.e. the user utterance, e.g. `userUtterance=I+want+to+book+a+trip`) to the received dialogue instance (e.g. `http://localhost:8080/nadia/engine/dialog/d1`) and display the answer, as shown in the response log in Figure 12.5.

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/plain
Transfer-Encoding: chunked

Where do you want to start?
```

Figure 12.5: Response log: retrieving the next system question

For detailed instructions on how to realise this test case with basic Linux commands, please refer to Appendix A.5. Apart from typing in your utterance and reading the response, the Web User Interface also comes with Automatic Speech Recognition (ASR)² and Text to Speech (TTS), both provided by Google. The links *show context* and *show dialogue* provide you with background information about the current system state and visualise the active dialogue definition.

12.2 Runtime Dialogue Model

On runtime we need more information than is specified in the dialogue description because we need to store the current state of the dialogue including the user's answers. This applies mainly for the ITOs because an ITO resembles a frame. It contains the following runtime information:

- the exact formulation of the system question, that has been used to ask for the corresponding information

²Google Chrome is needed to run the ASR

- the user's answer
- the interpreted value of the user answer
- a flag if the ITO has been visited
- a flag if the ITO has been filled

Further information about the state of the dialogue is stored in the dialogue manager's context. This includes the following details:

- has the dialogue already been started?
- is a question active?
- if true, a link to the current question
- a history of the dialogue turns
- a stack with the called tasks
- a time stamp of the last access

12.3 Dialogue Engine Architecture and Behaviour

The dialogue engine is the heart of the whole dialogue system and processes the dialogue definition that has been developed in this thesis. It makes use of an embedded Jetty web server and can be accessed over a REST interface. Alternatively, local console access is possible. If you want to deploy the dialogue system on an existing web server, you may want to use the provided war-file.

The dialogue engine basically works according to the standard IPO model. It takes a user utterance as input, processes it and returns a response as output. In the next section we will elaborate on the processing part.

If the loaded dialogue has not been started yet, the dialogue engine first identifies the start task and retrieves the first ITO, which is very often a greeting and an open-ended question. Otherwise, if the user sends an utterance to the system, it will get processed in two steps:

- identify dialogue act (SoDA)
- parse utterance and perform NLU

The dialogue act analysis is done before the natural language understanding in order to prevent the system from misinterpreting the user utterance. If dialogue acts are activated, the user utterance will only be interpreted as new information, if it has been recognised as *information providing*. Otherwise, it is treated as a request for a different task.

If the utterance is recognised as *information providing* and the NLU module is not able to process it within the context of the current question, the dialogue engine looks for alternatives:

- check if the utterance could be a correction
- check if the utterance is an answer to a different question from the same task
- check if a different task may be responsible and if so interpret the utterance in that task
- repeat the question if an interpretation is impossible

When all information is retrieved, the dialogue engine executes the action. Otherwise the next question will be asked and the check for completeness will be repeated. After the action has finally been executed, the dialogue engine determines what to do next based on the results of the action. This may lead to the redirection to a different task or to a follow-up question. Follow-up questions can be used to ask the user what he wishes to do next (e.g. “*Would you like to book another trip?*”).

12.3.1 Parsing and Natural Language Understanding

The analysis of natural language is more complicated than the mere detection of keywords. Although we have seen that with the help of dialogue acts and granular settings of the initiative we can improve the quality of dialogues, only a more sophisticated interpretation allows us to engage in a more natural dialogue. When only using a gazetteer, telling the system that you want to “*book a trip to Edinburgh*” may end up with a *departure city* of Edinburgh as this may be the first matching slot for a city name. Also the context (in this case the word *to*) needs to be analysed in order to correctly interpret the utterance as a reference to the *destination city*.

Although the development of NLU-modules is not part of the thesis, our model needs to provide an interface to this integral part of any dialogue system. The context of a keyword has already been identified as an important source of information in order to reduce ambiguity. Moreover, the context of the answer – i.e. the question – is a significant feature towards its interpretation, as can be seen in the following example where the same answer carries different meanings and is only interpretable in the context of the question.

Dialogue 12.42:

S: *Where do you want to go?*

U: *San Francisco*

S: *Where do you want to start?*

U: *San Francisco*

S: *What city do you want to know the weather of?*

U: *San Francisco*

As described in the last section, the dialogue engine first tries to interpret the answer in the context of the current question. Only if that fails, the context is stepwise relaxed to other questions and even to different tasks. However, depending on the dialogue

settings, after a successful comprehension, the question can be relaxed anyway in order to allow over-informative answers like “*I want to go to San Francisco on 21 May*”. The first interpretation of the answer is done via the active (or by the dialogue manager selected) ITO. Every ITO has an associated type that serves as the interface between the dialogue manager and the NLU component, i.e. the answer to a question of the type `fact.temporal.date` will be analysed by a NLU-module that is registered with the dialogue manager to be able to process answers of that very type. The same goes for all the other types of the proposed AQD hierarchy. If the NLU-module has identified the answer as matching (i.e. if a keyword that satisfies the type has been found), in a second step the context has to be analysed in order to get the meaning of the utterance and check if the answer also matches on a semantic and pragmatic level. The following examples give a brief understanding why this may be problematic.

Dialogue 12.43:

S: *Where do you want to start?*

U: *I'd like to go to **San Francisco**.*

S: *Where do you want to start?*

U: *I'd like to go from **Hamburg** to **San Francisco**.*

S: *Where do you want to start?*

U: *Somewhere in Germany, but not **Berlin**.*

S: *Where do you want to start?*

U: *I'd like to start in **Hamburg**, no wait, in **Berlin**.*

S: *Where do you want to start?*

U: *How is the weather in **Hamburg**?*

While the last ambiguity is handled by the dialogue act recognition, the other answers need to be disambiguated by the NLU module. The following example is even more complicated, because the dialogue system needs to recognise the task switch and resolve the anaphora.

Dialogue 12.44:

S: *When do you want to go to San Francisco?*

U: *How is the weather **there**?*

Also corrections can be the cause of misunderstandings, as can be seen in the following example. Consequently, a NLU-module should identify that the word *two* does not refer to the age of the child as a simple keyword-spotting approach would assume.

Dialogue 12.45:

S: *How old is your child?*

U: *I'd like to travel with **two** children.*

Sometimes, also descriptions or paraphrases of facts need to be interpreted. Instead of answering with *two*, in the next example the user decides to list the persons who will take part in the trip.

Dialogue 12.46:

S: *For how many persons do you want to book a trip?*

U: *For me and my wife.*

Referring to the features that we have identified in Section 3.2.4 as characteristic and important for natural dialogues, we can also list some of them as challenges in understanding natural language utterances, as we have seen in the preceding dialogues:

- over-informativeness
- negations
- corrections
- discourse and anaphora
- colloquial language
- contextual relations (e.g. temporal and local expressions)

For solving this problem, we can think of different approaches that support NLU, e.g.

- Named entity recognition
- POS tagging
- Shallow or deep parsing / chunking
- N-grams
- Gazetteers
- Stemming
- Pattern Matching

The analysis of suitable processing methods and the development of NLU modules has been addressed by Fanter (2014). In particular, he developed understanding modules for *locations*, *dates and times*, *quantities* and *decisions*. All the understanding modules share a common preprocessing that consists of several different steps that are illustrated in Figure 12.6. After the input text has been split up into sentences and has been tokenised, the tokens are lemmatised and annotated with POS tags. Now chunks and

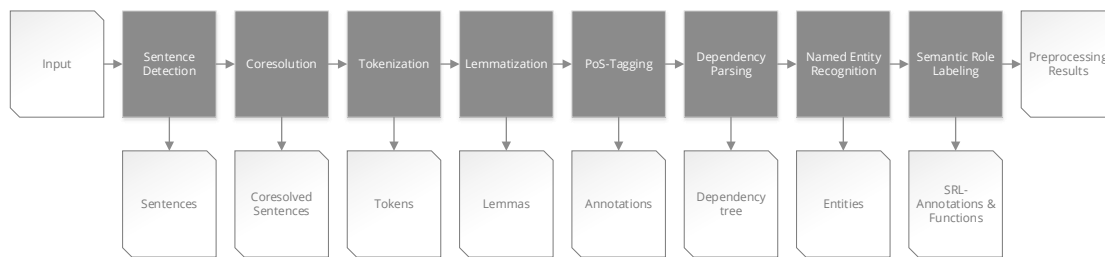


Figure 12.6: Preprocessing (Fanter, 2014)

dependencies are extracted which is a prerequisite for semantic role labelling. Now a named entity recognition is performed and coreferences are identified (Fanter, 2014).

As a result, we have a structured and labelled user utterance that is enriched with various information that we need for the rest of the understanding process. In the next step, Fanter identifies the *executing verb*, i.e. the verb that is important for understanding the user’s intention (in contradiction to the auxiliary verb), and its arguments (valences) as you can see in Figure 12.7.

A0	I
A1-LOC	to
A1	London

Figure 12.7: Semantic Role Labelling: arguments for *travel* (Fanter, 2014)

Finally, negations are determined. Now the specific understanding modules are called by the NADIA system. Every understanding module has two modes: flat processing (if the answer only consists of keywords or short phrases) and deep processing (if the answer is a full sentence).

When flat processing is used, the *city understanding module* checks for prepositions to infer a direction (from/to). If no preposition is present, the pragmatic aspect from the AQD (e.g. end of trip) is used and it is assumed that the answer addresses this aspect (consider the answer “*Glasgow*” to the question “*Where do you want to go?*”). Afterwards a named entity recognition is performed and backed with a gazetteer lookup in connection with a Levenshtein distance to take care of variations in spelling. The deep processing also checks if the found entity is an argument of the *executing verb* (“*I’d like to go to Berlin because it is raining in Hamburg*”). Moreover, a list of prepositions for specific verbs is used to give more information about the referring direction.

The *decision understanding module* checks for answers of the type *yes* or *no*. What

seems like a simple lookup of phrases like “*of course*” or “*nope*” becomes more difficult if the user answers the question “*Do you like a drink?*” with “*A coke please*”, which means *yes* and must furthermore be handled like an over-informative answer. The interpretation of the answer as a positive one is done with the help of hyperonyms. If the user makes use of a hyponym of the noun in the question (coke \subset drink) or the AQD’s reference (e.g. beverage), than it is assumed (if the sentence is not negated) that we deal with a positive answer. The deep processor also takes care of answers that just reformulate the question (“*Do you want a drink*” – “*I want a drink!*”). This is realised by a comparison of the verb’s arguments in both the question and the answer. Furthermore, answers of the type “*I do*” are understood.

Finally, the *date understanding module* bases on a pattern matching approach. It has to be noted, that numerals like *thirteen* need to be converted into numbers (13) first. Moreover, the module is able to identify the role of a date, e.g. a birthday.

These improved language understanding modules seamlessly integrate with the NADIA system by using the provided interfaces. The successful usage of the AQD underlines that the dialogue model is reasonable and useful.

12.3.2 Actions

Actions connect the dialogue system with the real world. The utterances that have been given in natural language in several turns are combined to a request that aims at fulfilling the user’s demand. Because there are various types of back-ends, the dialogue system needs to provide a flexible and extendable way of addressing these. Currently the following action types are supported:

- Java class loader
- Groovy
- HTTP/REST (post, get)

With the *Java class loader* any compiled Java class that extends the *JavaAction*-class provided by the dialogue system can be loaded and executed. This allows the developer to execute any code but is the most effortful way of addressing a back-end as you have to write and compile external code. With *Groovy* it is possible to include script code directly into the dialogue definition without having to deal with external files. However, the easiest way of accessing web interfaces is by *REST*. The following XML extract shows how to access the *openweathermap.org* API by a get-request. The provided XPath is then applied to the retrieved XML data and returned to the user as **#result**, which may lead to an answer like “*The temperature in San Francisco is 28 degrees*”.

```
<action>
  <httpAction>
    <returnAnswer>true</returnAnswer>
    <utteranceTemplate>The temperature in %getWeatherCity is #result degrees.</
      utteranceTemplate>
    <method>get</method>
    <params>q=%getWeatherCity&mode=xml&units=metric</params>
```

```

<url>http://api.openweathermap.org/data/2.5/weather</url>
<xpath>/current/temperature/@value</xpath>
</httpAction>
</action>

```

Listing 12.1: HTTP Action

A more complex example includes a *resultMapping* that redirects to another task based on the value of the result. The example in Listing 12.2 is taken from our number guessing game. The mapping links to itself (the guessing task) until the correct number has been guessed. For this purpose, the value of the *result variable* is compared with the *result value*.

```

<action>
  <httpAction>
    <resultMappings>
      <resultMapping>
        <message>was too big.</message>
        <redirectToTask>guessNumber</redirectToTask>
        <resultValue>TOO_BIG</resultValue>
        <resultVarName>result</resultVarName>
      </resultMapping>
      <resultMapping>
        <message>was too small.</message>
        <redirectToTask>guessNumber</redirectToTask>
        <resultValue>TOO_SMALL</resultValue>
        <resultVarName>result</resultVarName>
      </resultMapping>
      <resultMapping>
        <message>it is! Congratulations!</message>
        <resultValue>CORRECT</resultValue>
        <resultVarName>result</resultVarName>
      </resultMapping>
    </resultMappings>
    <returnAnswer>true</returnAnswer>
    <utteranceTemplate>%getNumber</utteranceTemplate>
    <method>get</method>
    <params>guess=%getNumber</params>
    <url>http://mmt.et.hs-wismar.de:8080/NumberGuessing/NumberGuessing</url>
    <xpath>//code</xpath>
  </httpAction>
</action>

```

Listing 12.2: HTTP Action with result mapping

Whenever the action is executed, a system utterance will be created that consists of the global utterance template and the conditional *resultMapping*-message, e.g. “38 was too big”.

12.3.3 Utterance Templates

In several parts of the dialogue definition we need access to internal data. In Listing 12.1 you can see two different types of references in the *utterance template*. References that start with a percentage sign refer to the value that has been gathered by the corresponding ITO, while references starting with a hash refer to variables defined by the action (an `HttpAction` only defines `#result`). These references can also be used to construct the query parameters or to formulate system utterances.

12.3.4 Task Stacking

In order to realise a natural dialogue, it is necessary to support subdialogues. Often, a user spontaneously ignores a question and asks a question himself without the wish to abort the first dialogue. As can be seen in the following example, two independent tasks are called within one dialogue.

Dialogue 12.47:

S: *Where do you want to go?*
U: *How is the weather in Riga?*
S: *25°C and sunny.*
S: *Where do you want to go?*
U: *OK, to Riga please.*

The dialogue engine recognises the user's wish to address a new task with the help of the dialogue act recogniser, keeps track of the current task and loads the new task. Once the new task has been completed (i.e. all questions have been answered and the action has been executed), it is removed from the stack and the previous task is continued. The whole dialogue is finished if all tasks on the stack are completed, i.e. if the stack is empty. In order to realise an endless dialogue loop, we can use an open-ended question and associate it with an NLU module that only listens for keywords that abort the task (like "bye"). This means the task can only be removed from the stack if the user says good bye and ends the dialogue.

Dialogue 12.48:

S: *How may I help you?*
U: *How is the weather in Riga?*
...
S: *How may I help you?*
U: *I'd like to book a trip.*
...
S: *How may I help you?*
U: *Bye.*
S: *Good bye!*

In any other case, the NLU component cannot interpret the user utterance so that it gets redirected to one of the remaining tasks as part of the backup strategy. Whenever the selected (backup) task is finished, the task with the open question is activated again, because it is still on the stack as it has not been successfully answered yet. By this means, the question "How may I help you" reoccurs until the user wishes to end the dialogue, as you can see in Dialogue 12.48. This allows the design of a dialogue that always comes back to an open question if no other task is active.

12.4 Dialogue Development Environment

One of the factors why many dialogues do not seem very human and are therefore regarded as not user-friendly is that the development of natural dialogues is too complex and expensive. That's why we have to find ways to facilitate the creation of natural dialogues. Until now, we have described a dialogue model and a dialogue engine that is able to process this model and interact with the user with the help of different user interfaces. We have separated the dialogue engine from the description of the dialogue's information demand. The dialogue designer now needs to create a dialogue specification that bases on this model. This is the basis for facilitating the development of dialogues and making use of the methods that have been developed in this thesis.

The described dialogue model is XML-based and can be written manually in any standard text- or XML-editor. The provided schema file (see Appendix A.4) makes code completion possible and allows syntax checks. However, we recommend the usage of the provided Java library for creating the dialogue, as this requires less code. The Java-based model can then be converted into XML by usage of the method `Dialog.toXML()`. Of course, the system can also directly process the Java-written dialogue model (without the XML-conversion). However, this possibility can only be used if the developer has direct access to the source code of the dialogue system, which is in general not the case and only an option while debugging the engine.

To give you an impression on how to create a dialogue model with Java, refer to the following source code extract.

```

1 Dialog dialog = new Dialog("example");
2 dialog.setGlobal_politeness(2);
3 dialog.setGlobal_formality(2);
4 dialog.setStart_task_name("getTripInformation");

6 Task task1=new Task("getTripInformation");
7 bagOfWords = new ArrayList<String>(Arrays.asList("travel", "book", "journey", "
    trip"));
8 task1.setSelector(new BagOfWordsTaskSelector(bagOfWords));

10 ITO ito=new ITO("getDepartureCity", "Where do you want to start?", true);
11 task1.addITO(ito);
12 AQD aqd=new AQD(new AQDType("fact.named_entity.non_animated.location.city"), new
    AQDContext("begin", "trip"), new AQDForm());
13 ito.setAQD(aqd);
14 ...

```

Listing 12.3: Java code to define a dialogue model

Here, we design an example dialogue that defines a task for getting trip information that gets activated if the user mentions one of the keywords *travel*, *book*, *journey* or *trip*. The first question (ITO) asks for the departure city and makes use of language generation to generate a suitable formulation for a medium politeness and formality. The answer type (AQD Type) realises the connection to the corresponding natural language understanding module. As you can see, the definition of the actual behaviour is implemented in the dialogue engine and just controlled by parameters. In this way, the developer saves a lot of work and does not need to bother with the implementation of subdialogues, mixed

initiative, the definition of grammars or language understanding methods.

Because both methods (Java, XML) still require a certain level of programming skills, we aim at creating a development environment that allows the (non IT) dialogue designer to create a dialogue with the help of a graphical dialogue development environment (DDE). The DDE supports the designer in creating valid dialogue models. All the necessary features are created automatically and the user is just asked to fill in the corresponding values (e.g. the answer type of a question). An object library allows the user to add additional elements, e.g. more ITOs or tasks, which is necessary to create a complete dialogue. Finally, the resulting model can be saved as an XML file or can be directly loaded into the NADIA dialogue engine in order to run and test the dialogue.

The heart of the DDE is the schema that has been introduced in Figure 11.2. Depending on the element type, respective GUI elements are rendered. A boolean element can be displayed as a check box while a string type requires a text input field. If the parser encounters a complex element, it has to be analysed until the deepest level (that defines simple types) is reached. In order to guarantee a valid dialogue model, we must not only regard the structure of the elements, but also regard its contents, i.e. we need to define all the possible values that an element supports: the type information `string` for the parameter *dialogue act* is not sufficient. Additionally, we need to provide the system with the information that *information seeking*, *information providing* and *action requesting* are valid values for these elements. This can be done with a mapping of the elements to a list of all possible values or a formal description of them (e.g. regular expressions).

The implementation of the DDE has been realised in the Master's Thesis by Sötebier (2013) which "*investigates the conception and implementation of a web-based development environment*" for dialogues and includes the following features:

- web-based (no installation needed, platform-independent)
- visual creation of the dialogue definition without the need to have programming skills
- supports the user in creating only valid dialogues by ensuring that only valid values can be entered
- multilingual graphical user interface
- version management (conversion of the loaded model into the latest version of the dialogue model)
- integrated NADIA access to test the dialogue

The resulting DDE – which has been implemented using Eclipse RAP – "*dynamically generates a user interface by interpreting an XML-schema and supports the user in creating a dialogue description file, which serves as the input for the dialogue engine.*" Also, "*aspects of version management, validation and the restriction of user input*" are examined.

Figure 12.8 illustrates the DDE and three exemplary steps of designing a dialogue using drag and drop and a tree-based user interface. First, you create a new dialogue object

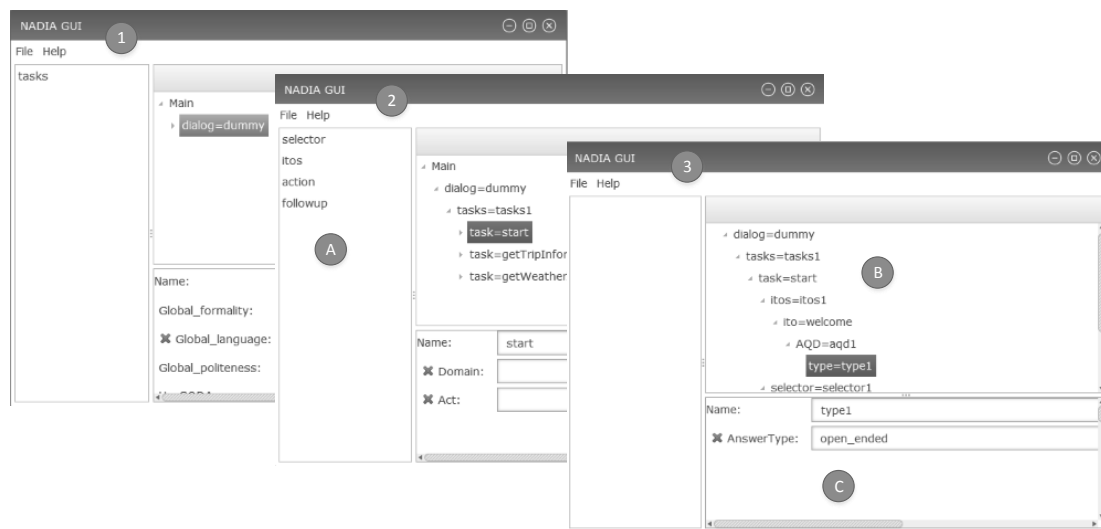


Figure 12.8: Dialogue Development Environment

(1). Then you add a new task in the second step (2). In the third step, you can see how the designer added an ITO and an AQD (3). On the left hand side of the window (A) you can select all the possible complex child elements, e.g. if you are on the dialogue level, you may add tasks or if you are on the task level, you may add selectors, ITOs or actions. The middle view (B) shows you the current structure of your dialogue and allows you to activate the third view on the bottom (C), that enables the designer to change the values for the simple elements of the current complex element (e.g. the name of the dialogue or the answer type of an AQD). Here, you can also enable or disable features that control the dialogue behaviour (e.g. mixed initiative by activating over-informative answers and enabling the shift of tasks). When the designer has completed the dialogue, it can be tested by uploading it to the NADIA engine and calling the NADIA web interface that we have described in Section 12.1. This is done automatically when calling *Load XML File into NADIA* from the main menu.

12.5 Summary

In this chapter we have demonstrated how the dialogue model can be applied and processed. For this purpose we have developed the NADIA dialogue system, which is a dialogue engine that is able to process the dialogue model that we have developed throughout the last chapters. We have described the architecture, communication and user-interface and have created exemplary dialogues. Hereby, we have shown that the model works and is applicable in real scenarios. One of our aims is the facilitation of the creation of natural dialogues. By the use of NADIA, we demonstrate how easy it is to design

dialogues without needing to bother with programming the dialogue engine or implementing subdialogues or mixed initiative (further details in the next chapter). However, development can be further simplified by offering a graphical development environment that ensures the design of valid dialogue models without the need to have knowledge in programming languages.

Moreover, with the two Master's Theses with respect to NADIA, we also underline the potential of this software and the relevance of the topic in general. Because NADIA should also be usable during university courses, we chose a completely web-based approach. Hence, the software does not need to be installed on every workstation and is independent from the operating system or software policy. Alternatively, NADIA can be started locally within an embedded web server which is also very straightforward and puts only minimal requirements on the environment. The server component can be accessed by simple HTTP calls (post and get) which allows an easy development of clients (e.g. webpage or instant messenger).

Part IV

Scenarios and Evaluation

Synopsis

In this final part we present use cases for the developed dialogue model and show how to implement these. Afterwards, we conduct an external experts assessment and evaluate the results. Then, we summarise the results of a user study where we have asked participants to realise given dialogue scenarios with our model and let them rate the effectiveness and simplicity of this development approach. Finally, we conclude with an analysis of existing limitations that give room for future work and summarise the results of this thesis.

As far as the customer is concerned,
the interface is the product.

(Jef Raskin, 1943 – 2005)

13

Use Cases

We now describe some example dialogues that are being modelled subsequently as use cases in order to prove the operationality of our approach. We distinguish two types of dialogue systems: control systems and information systems. While systems of the first category are used to control applications and devices which leads to only a few questions from the system (only in case of errors or ambiguities), systems of the second category ask a lot of questions in order to get all the information they need for being able to formulate a query and present the results. We provide use cases for both types of systems.

13.1 Room Control Scenario

As a first step, we create a simple simulated light bulb back-end, where the user is only able to switch the light on and off. The main focus of this use case is the application of the *SoDA* which allows us to differentiate an information-seeking request from an action request. For this use case, we need to create two different tasks: `getLightbulb` and `setLightbulb`. Both are activated by the same *bag of words selector* that listens for the words *bulb* and *switch*. Here, we need the user's intention, i.e. the results of the SODA recognition, in order to select the correct task. Hence, we define the task `getLightbulb` as *information-seeking* and `setLightbulb` as *action-requesting*. The resulting visualisation of the model can be seen in Figure 13.1 and may result in the following dialogue:

Dialogue 13.49:

U: *I'd like to know if the light is switched on.*

S: *The light is switched off.*

U: *Okay, then switch the light on please!*

S: *The light has been switched on.*

```
INFO (DialogManager): init dialogue
INFO (DialogManager): switch task to: start
INFO (DialogManager): getting next question
INFO (MaximumEntropyModel): Utterance: "I'd like to know if the light is
switched on" resulted in: seek[0,5002] action[0,4905] prov[0,0093]
-> seek
INFO (DialogManager): Could not be interpreted in current task: start
```



Figure 13.1: Visualisation of the light bulb tasks

```

INFO (DialogManager): responsible task is: getLightbulb
INFO (DialogManager): switch task to: getLightbulb
INFO (DialogManager): frame filled , executing action
INFO (DialogManager): getting next question...
INFO (DialogManager): no more questions , task completed
INFO (DialogManager): switch task to: start
INFO (DialogManager): getting next question
INFO (MaximumEntropyModel): Utterance: "Okay, then switch the light on
please" resulted in: seek [0,0401]  action [0,9420]  prov [0,0179] ->
action
INFO (DialogManager): Could not be interpreted in current task: start
INFO (DialogManager): responsible task is: setLightbulb
INFO (DialogManager): switch task to: setLightbulb
INFO (DialogManager): looking for answers
INFO (DialogManager): a parser matched for ITO getLightAction: OnOffParser
matched sequence 26-28 (on) as onoff:on
INFO (DialogManager): frame filled , executing action
INFO (DialogManager): getting next question

```

Listing 13.1: Log extract: Dialogue Manager

The log extract in Listing 13.1 demonstrates how the task selection and SoDA recognition works. Interestingly, the `getLightbulb` task has no ITO at all because all the necessary information has already been specified by calling this very task. However, if

there exist several light bulbs or in general several devices that can be switched, the task would include an ITO that asks for the intended device. When we take a look at the actions, we observe another difference: switching the bulb obviously triggers a post request with the new state as parameter, while reading the status leads to a get-request.

13.2 Weather Information System

In the first use case we have demonstrated a dialogue where the user utters a command that leads to changes of the state of real world objects. This has been simulated by a proprietary back-end. In this use case we specify an information-seeking dialogue that calls the *openweathermap*-API and results in a dialogue like this:

Dialogue 13.50:

S: *How may I help you?*
U: *How is the weather in Aberdeen today?*
S: *The temperature in Aberdeen is 12°C.*
S: *How may I help you?*
U: *And how is the weather in Edinburgh?*
S: *The temperature in Edinburgh is 14°C.*

As always, we can use plain XML or create the dialogue description with the help of the Java library. In Java we write the following code¹:

```
1 Task task=new Task("getWeatherInformation");
2 bagOfWords = new ArrayList<String>(Arrays.asList("weather",
3     "temperature"));
4 task.setSelector(new BagOfWordsTaskSelector(bagOfWords));

6 ito=new ITO("getWeatherCity", "For what city do you want to know the
7     weather?", false);
8 aqd=new AQD();
9 aqd.setType(new AQDType("fact.named_entity.non_animated.location.city"));
10 ito.setAQD(aqd);
11 task.addITO(ito);

12 HTTPAction httpaction=new HTTPAction("The temperature in %getWeatherCity
13     is #result degrees.");
14 httpaction.setUrl("http://api.openweathermap.org/data/2.5/weather");
15 httpaction.setMethod("get");
16 httpaction.setParams("q=%getWeatherCity&mode=xml&units=metric");
17 httpaction.setXpath("/current/temperature/@value");
18 task.setAction(httpaction);

19 dialog.addTask(task);
```

Listing 13.2: Weather task definition

We first define a new task which we connect with a bag of words selector. We then specify an ITO that asks for the name of the city that we want to know the weather of.

¹We illustrate the examples with Java code because this is more compact

Afterwards, the HTTP action is used to perform a get request on the *Openweathermap API* by using the city name and to extract the temperature from the result set with the help of the XPath. In contrast to the last example, we definitely need an ITO in order to be able to formulate the search request. However, it is also possible to skip this extra dialogue step by including this information into the task-selection utterance (given that over-informative answers are enabled) like this: *“How is the weather in Edinburgh?”*. A more verbose version of this dialogue would look like this:

Dialogue 13.51:

U: *I'd like to know the weather.*

S: *For what city do you want to know the weather?*

U: *Edinburgh.*

13.3 Trip Information System

We now want to design a price information system for trips. This requires asking for several information before being able to execute an action, i.e. departure city, destination city, number of persons, start date and return date. This means we need five ITOs. At this point, we also want to demonstrate the possibility to write a Groovy script and therefore only simulate the back-end (here, we just make up a random price).

```
GroovyAction gaction = new GroovyAction("Okay, the cheapest flight from %  
getDepartureCity to %getDestinationCity will be #price Euros.");  
gaction.setCode("Random r=new Random(); Integer p=r.nextInt(300)+100;  
executionResults.put(\" price\",p.toString());");
```

Moreover, we make use of the language generation module. Instead of specifying an utterance like *“Can you please tell me where you want to go?”* we just define an AQD:

```
aqd=new AQD(new AQDType("fact.named_entity.non_animated.location.city"),  
new AQDContext("end","trip"), new AQDForm());
```

If we leave the *form* empty, the global dialogue settings will be used instead. This definition leads to a dialogue like this one:

Dialogue 13.52:

S: *How may I help you?*

U: *I'd like to get price information about a flight from Hamburg to Edinburgh.*

S: *When do you want to depart?*

U: *On 1 July.*

S: *And when do you want to come back?*

U: *On 14 July.*

S: *For how many persons?*

U: *2.*

S: *Okay, the cheapest flight from Hamburg to Edinburgh will be 320 €.*

U: *Thank you! Bye!*

S: *Bye!*

If the user does not provide necessary information on his own, the system asks for it:

Dialogue 13.53:

- S: *How may I help you?*
U: *I'd like to get price information about a flight.*
S: *Where do you want to depart?*
U: *Hamburg.*
S: *Can you please tell me where you want to go?*
U: *Edinburgh.*
S: ...

This behaviour is one of the most important characteristics of mixed initiative and can be changed in the dialogue settings (allow to switch tasks, allow over-informative answers, allow to ask a different question, allow corrections).

13.4 Multiple Tasks: City Information System

Until now, we have only described single tasks. We now want to connect several tasks and create a more interesting dialogue that is able to switch between different topics and address several back-ends. In the following example, you can see a dialogue that accesses *Wikipedia* for city information, *Openweather* for the current temperatures and *Google Maps* to get the distance between two cities.

Dialogue 13.54:

- S: *How may I help you?*
U: *Can you give me some information about Edinburgh?*
S: *Edinburgh is the capital city of Scotland, situated on the southern shore of the Firth of Forth. With a population of 482,640 in 2012, it is the largest settlement in Lothian and lies at the centre of a larger urban zone of approximately 850,000 people.*
U: *Sounds interesting. How is the weather in Edinburgh?*
S: *16°C*
U: *Okay, and how far away is Edinburgh from Hamburg?*
S: *Edinburgh is 1560 km away from Hamburg if you go by car.*
U: *Thanks for the information! Bye.*

Moreover, the existence of several tasks introduces the possibility to call a task while another task has not been finished yet. If we combine the trip information system from the last section with these other tasks, we may end up with the following dialogue:

Dialogue 13.55:

- S: *How may I help you?*
U: *I'd like to get price information about a flight.*
S: *Where do you want to depart?*
U: *Hamburg.*

S: *Can you please tell me where you want to go?*
 U: *Can you tell me something about Edinburgh?*
 S: *Edinburgh is the capital city of Scotland, situated on the southern shore of the Firth of Forth. With a population of 482,640 in 2012, it is the largest settlement in Lothian and lies at the centre of a larger urban zone of approximately 850,000 people. Where do you want to go?*
 U: *How is the weather in Edinburgh then?*
 S: *16°C. Can you tell me where you want to go?*
 U: *Okay, I want to go to Edinburgh.*
 S: *When do you want to depart?*
 U: ...

The dialogue designer does not need to change anything in the dialogue definition. He just needs to specify the information demand by defining the tasks and ITOs and make sure that the task-switching and SODA properties are enabled (default).

13.5 Acting upon the Results: Number Guessing Game

Often, the scenario requires the system to react on the results. This is the case, if we don't pursue a mere informational dialogue, but want to iteratively work on a task together with the user. We want to clarify this by the well-known number guessing game. As always, we first need to create an ITO that asks for the number that the user thinks the system has made up.

```
Task task = new Task("guessNumber");
...
ito=new ITO("getNumber", "Guess a number between 1 and 99!",false);
task.addITO(ito);
aqd=new AQD();
aqd.setType(new AQDType("fact.quantity"));
ito.setAQD(aqd);
...
```

We then define an HTTPAction that addresses our back-end logic. Unlike before, we now add a *result mapping* that analyses the result value and activates the given task, e.g. if the value of the variable *result* equals *TOO_BIG*, the task *guessNumber* will be (re)activated. In this case, we have created a loop with the break condition *result==CORRECT*.

```
httpaction.setResultMapping(new ActionResultMapping("result","TOO_BIG","
    was too big. ","guessNumber"));
httpaction.setResultMapping(new ActionResultMapping("result","TOO_SMALL","
    was too small. ","guessNumber"));
httpaction.setResultMapping(new ActionResultMapping("result","CORRECT","it
    is! Congratulations!",null));
task.setAction(httpaction);
```

Another feature is the *follow-up question*. It can be used if an action has been executed and has not called another task. In that case, we can ask a question that indicates what

the user wants to do next. In our example, we ask the user if he wants to play another round of the game.

```
followupito=new ITO("playAgain", "Do you want to play again?",false);
aqd=new AQD();
aqd.setType(new AQDType("decision"));
followupito.setAQD(aqd);
followup=new FollowUp();
followup.setIto(followupito);
answerMapping=new HashMap<String, String>();
answerMapping.put("YES", "guessNumber");
followup.setAnswerMapping(answerMapping);
task.setFollowup(followup);
...
```

The definition of a follow-up question is straightforward as it is just an ITO that is connected with an answer mapping similar to that of the action. By this means, we just reuse an existing part of the model and extend it with information that specifies the task that the user will be redirected to depending on his answer.

These use cases have demonstrated the capabilities of the model. By loading these dialogues into NADIA, we have also proven that the dialogue engine works and that the model is sensible, flexible and can be easily used to create different dialogues. Furthermore, through the separation of the dialogue engine from the dialogue specification, we can efficiently create new dialogues without having to bother with the development of the dialogue engine. In the next chapter we conduct an external evaluation.

No amount of experimentation can ever prove me right;
a single experiment can prove me wrong.

(Albert Einstein, 1879 – 1955)

14

Evaluation

In this chapter we compare our results with existing standards that are used in industry and let experts and untrained users validate the system to analyse its strengths and weaknesses.

14.1 Comparison with Voice XML

Voice XML is a language to create interactive voice response systems, i.e. speech dialogue systems that are optimised for the phone. It can be compared to what HTML is for the Web. The main elements are *menu* and *form*. A menu consists of different *choices* and can be used to transfer the user to another URI, i.e. another dialogue. In NADIA, we can use an open question instead, because every task has the knowledge to recognise that it is responsible for the utterance. Hence, we don't need explicit choices. However, these can be modelled with a follow-up ITO (*“Do you want to know the weather or want to get stock prices?”*, type: `item(weather,stock)`) if necessary. A form consists of *fields* that represent pieces of information of the whole dialogue. Every field can contain a *prompt* that outputs a message, a *grammar* that specifies the words for the recogniser and optional events like *no input* or *no match*. However, the most important one is the *filled-event* that specifies how to react when the user has answered some or all questions. A field is similar to an *ITO*, while a form resembles a *task*.

Although both approaches, Voice XML and NADIA, are similar, there are some differences: While Voice XML features IVR-related capabilities, NADIA focusses on special aspects of dialogue research as can be seen in Table 14.1. With Voice XML it is possible to react on ASR events like *no input* and *no match*, to use audio files instead of speech synthesis or to make use of DTMF instead of speech recognition. However, the main difference is the powerful inline scripting that enables the dialogue designer to control the dialogue behaviour and to define the reaction.

The focus of NADIA, on the other hand, is not the definition of the dialogue flow, but the modelling of the information demand and the reactions of a dialogue. The behaviour should be realised by the dialogue manager on the fly. The dialogue designer only specifies some general features like the intended type of initiative, the preferred linguistic style and the data type of the answer. This requires the use of language generation methods that

Feature	Voice XML	Nadia
ASR events	+	-
inline scripting	+	-
create variables	+	-
audio files	+	-
call transfer	+	-
DTMF	+	-
mixed initiative	+	+
back-end integration	-	+
dialogue acts	-	+
data types / NLU	-	+
language generation	-	+
automatic task switching	-	+

Table 14.1: Voice XML vs. Nadia

are able to create utterances based on abstract concepts. Another difference is NADIA's direct back-end integration. We provide REST- and Java connectors as well as inline Groovy scripting. In Appendix A.6 we show a short implementation of a calculator in both, NADIA and Voice XML. Voice XML requires an application server that dynamically creates Voice XML code based on a proprietary dialogue description (comparable to ASP or JSP which both create HTML), e.g. *Voxeo Voice Objects*, in order to access an external system and react on the results. Moreover, NADIA provides a set of predefined NLU modules that can be addressed by a type hierarchy. This is an advancement of built-in grammars that some media platforms provide. Also, dialogue acts are used to ease NLU and to prevent the confusion of answers and requests. Mixed initiative is supported by both approaches, although NADIA supports a more granular control of the different characteristics of mixed initiative. Another difference is NADIA's task switching facility that can be controlled by simple selectors. By this means, it is possible to invoke any other dialogue as a subdialogue without manual reference. Please note that this feature differs from the *subdialog*-element in Voice XML, whose only intention is the reuse of code, e.g. a confirmation that is explicitly referenced within a form.

14.2 Comparison with AIML

The Artificial Intelligence Markup Language (AIML) is an XML-based language for the creation of chatbots. It is based on Weizenbaum's Eliza and consists of the two primary elements *pattern* and *template* which are both enclosed by a *category*. The pattern refers to the user's input and the template defines the system's answer. Variables allow the storage and recalling of information in order to remember the user's name or any other

answer that he has given. Pattern wildcards make the dialogue more robust in that it is not necessary to always use the exact phrasing that has been specified.

The main principle of this approach is just a huge collection of user questions and possible system answers. Often, the system just responds with a psychiatrist-like counterquestion (“*How does this make you feel?*”) that should hide the system’s inability to interpret the user’s concern. By the use of references (**that**), system utterances can be conditioned on the user’s input and the last system utterance (if the last question was x and the user said y then say z). The **srai**-element allows the redirection of templates so that different patterns can be mapped to the same response.

The most severe drawback is the missing possibility of accessing back-ends. All the knowledge must be defined within the dialogue and no external data sources can be accessed. However, there are two possible solutions: the introduction of custom tags, which requires a proprietary interpreter, or waiting for AIML 2.0¹ which will allow communication with web services.

As well as Voice XML, AIML does not offer any dialogue research oriented features like dialogue acts or language generation. Moreover, it is not possible to influence the dialogue behaviour or to use advanced natural language understanding methods.

14.3 Expert Interviews

In order to prove that our model addresses an actual demand and that the approach is sensible and useful, we have asked seven experts from both academia and industry about the resulting NADIA system (that implements the theory of this thesis). Consequently, the interviews do not directly refer to the theoretical background but to the application and usage of the model. The participants were presented with the NADIA software, a short description of the intention of NADIA, and a manual that describes how to use it, i.e. how to create a dialogue. Every participant had to give a short assessment (free text) and had to evaluate novelty, demand, ease of use, level of simplification of the development, and level of contribution towards the creation of more natural dialogues on a Likert scale from one to five. You can find the results in Appendix A.7. We now provide an overview of the results.

The experts gave an average score of 4.1 out of 5, which attests the system a very good quality. The questions of the interview are divided into idea and realisation. The experts scored the novelty of the approach with 4.3 and the demand with 4.4 out of 5 points. This confirms that our idea is innovative and needed, i.e. our model addresses a real demand in a new way.

The second part of the questions relates to the NADIA software and assesses the application of the model (usability and functionality) which allows us to draw inferences about the viability of the theoretical background. The experts state that the model is easy to use and that it facilitates the development of speech dialogue systems. Although the model is fully functional and an improvement in comparison to existing approaches, the

¹<http://alicebot.blogspot.de/2013/01/aiml-20-draft-specification-released.html>

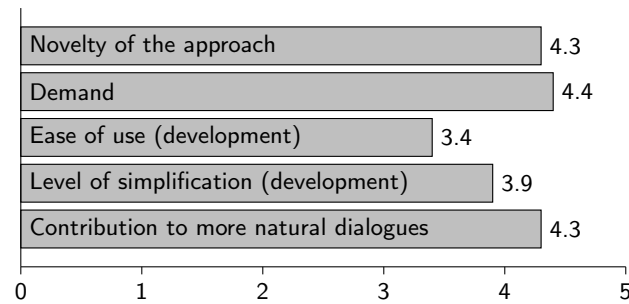


Figure 14.1: NADIA expert survey: mean scores

creation of dialogues that base on this model can still be further facilitated as the scores of 3.4 and 3.9 points indicate. To find possible aspects for improvement, we distinguish between three layers of facilitation:

Development of a dialogue model: A dialogue model is the basis for reusability and allows the separation of the dialogue behaviour’s implementation and the specification of the dialogue itself. (*solved*)

Development of a dialogue engine: A dialogue engine that processes this model facilitates the creation of dialogues because we can realise different dialogues with the same engine and any dialogue profits from improvements of the engine. Moreover, due to the separation of dialogue specification and dialogue engine, the developer just needs to specify the dialogue instead of reimplementing natural language understanding modules or different dialogue strategies again and again. (*solved*)

Creating dialogues based on the model: The specification of a dialogue based on the model still requires some basic programming skills because we need to create an XML file that will be loaded into the engine. Although the mere existence of such a model is a strong facilitation, the creation of dialogues can still be facilitated by a dedicated development environment. (*solved, but not part of the evaluation*)

Referring to Figure 14.1, we can see that the demand for such a system, the approach itself and the contribution towards natural dialogues have been rated very high. However, the application of the model, i.e. the dialogue development process, can be further simplified. While the first two of the just named issues have been addressed in this thesis, the third one has a practical background and has been solved by Sötebier (2013), who has created a visual development environment for NADIA which releases the end user (e.g. the dialogue designer) from using Java or XML to create the dialogue. We assume that this would further improve the usability scores if we ran the evaluation again.

The experts assess the system very positively and emphasise on different aspects. Ian O’Neill focusses on the ability to use subdialogues and states that:

“Nadia deals very effectively with user-driven shifts in dialogue context. In this respect it makes a very valuable contribution to development of dialogue systems [...]. The challenges presented by both the modelling and management of mixed-initiative dialogue are very considerable, even to the dialogue specialist.”

Amy Isard highlights the integration of natural language generation methods. She explains that the system *“allows non-expert users to benefit from natural language generation technology to express the same content in multiple ways depending on the dialogue context”*. Moreover, she approves that the model allows *“to specify the structure of dialogues more fully than in existing freely-available systems”* which *“makes the process of designing dialogues in a new area much faster and simpler”*. This is also confirmed by Flavius Frasinca who underlines the importance of the easy configuration of the system. He states that *“one of the most innovative aspects of the dialogue model is its [generality] and ability to specialize by means of configurations. For example, the granularity level of the dialogue strategies [allows] one to have more control on the dialogue structure than by using existing, state-of-the-art, systems”*. Finally, Ashwin Ittoo affirms the need for such a system by pointing out the *“lack of [an] easy to use dialogue system”* and claiming that NADIA *“presents several strengths compared to more traditional tools like VoiceXML”*.

14.4 Usability Test

In addition to interviewing experts, we also asked six candidates (mostly computer science students) with a basic background in speech technologies to read the manual, create some tasks and evaluate the development process as well as the resulting dialogue. The participants were given similar questions as the experts. However, they were not as free in the evaluation as the experts, as they had to realise given tasks and evaluate the results from two different points of view:

- dialogue developer
- dialogue user

In particular, they had to indicate how easy the model is to use (dialogue specification in both XML and Java), i.e. how satisfied they were with the development process and how satisfied they were with the results (how usable is the resulting dialogue?).

Task Description

The participants were asked to create a dialogue that consists of different tasks. They were free to choose between the provided Java library or plain XML in order to create the dialogue. The first task should access the *Openweathermap API* and retrieve the temperature in a city that has been given by the user. Hence, this task consists of one *ITO* and an *HTTPAction*. Afterwards, the dialogue has to be extended by another task. The *Google Directions API* is used to get the shortest time between two cities (by car).

This includes two questions, i.e. two ITOs. Now, an open-ended question should be introduced for being able to switch between these tasks. Finally, a third task, that asks for the prices of flights, has to be implemented. The system questions are supposed to be defined by an AQD to make use of the language generation facilities. The back-end can be simulated with an inline *Groovy* script. This results in a dialogue like this:

Dialogue 14.56:

S: *How may I help you?*
 U: *I'd like to book a trip*
 S: *Where do you want to start?*
 U: *In Edinburgh*
 S: *And where do you want to go?*
 U: *How is the weather in Inverness?*
 S: *20°C. And what is your destination?*
 U: *Umm, how long do I need from Edinburgh to Inverness?*
 S: *2:56 hrs. And where do you want to go?*
 U: *Ok, to Inverness then.*
 S: *When do you want to depart?*
 U: *I'd like to travel from 01/08/2013 until 14/08/2013.*
 S: *And with how many persons?*
 U: *3*
 S: *The cheapest flight is 500 Euro.*

Results

The mean scores of the survey results are illustrated in Figure 14.2. If you are interested in the filled surveys, please refer to Appendix A.8. Please note that an easier or better usage leads to higher scores. All participants state that the given tasks can be completely realised with the NADIA dialogue model. None of the participants had severe problems in creating the dialogue so that they are all nearly completely satisfied. Moreover, they all think that the creation of the dialogue is easier to accomplish than with Voice XML.

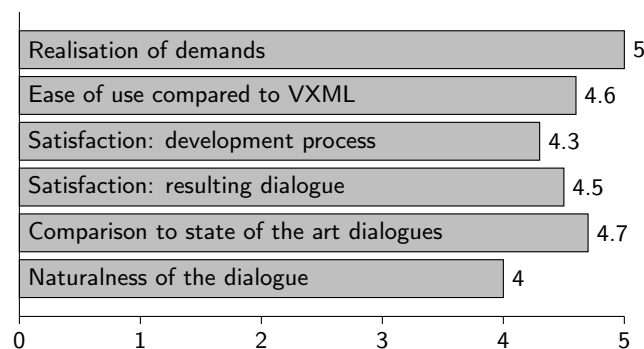


Figure 14.2: NADIA user study mean scores

When looking into the different features of NADIA, as displayed in Figure 14.3, we can identify which parts of the system may need improvement. We can see that back-end access, language generation (adaptive formulation) and language understanding seem to be more challenging than the other features. This is obvious because the understanding and generation of human language as well as the integration of back-ends naturally are complex tasks and require more details from the developer than the specification of the other features. While the realisation of subdialogues and mixed initiative does not require any additional information and instead can just be activated or deactivated, language generation requires the specification of the answer type (part of the AQD) and accessing a back-end expects the user to provide the URL and parameters of the service. Nevertheless, the participants give very positive scores for the different aspects of the development process.



Figure 14.3: NADIA user study mean scores: development aspects in detail

The results demonstrate that the development of a dialogue with NADIA is straightforward and easier than with common approaches like Voice XML. However, we also need to check if the resulting dialogue is functional and natural because the easy development of a dialogue that happens to be unnatural or unusable would render the aim of the thesis useless. When asking the participants about the dialogue, they are all very satisfied and rate it better and more natural than the dialogues of current systems that they came across in university or daily life.

The mean score over all questions is 4.6 out of 5 points, which makes a total percentage of 92%. Not a single question has been answered with a score below 3 points. The lowest average score for a question is 4 (*“How natural is the resulting dialogue?”*). We can conclude that the development of dialogues has been facilitated by NADIA and that the resulting dialogues are more natural than dialogues that emerge when interacting with current dialogue systems.

Apart from the survey, two of the participants also evaluated NADIA within a research seminar. In their report (Wendel and Thielbeer, 2013) they come to the conclusion² that *the clear, object-oriented creation of the dialogue in Java allows a fast implementation. Also, changes can easily be performed directly in the XML file.* The authors emphasise

²translated from German

the implementation of task shifts which makes the dialogue *more flexible and natural*. The dialogue designer only needs to define a task selector for every task and enable task switching in the global settings. Finally, Wendel and Thielbeer (2013) state that all the tasks could be completely realised with NADIA and *with the help of the supplied **actions**, it was possible to integrate external APIs and to create substantial querying systems*.

15

Analysis and Discussion

We use NADIA to indirectly evaluate the functionality of the model that we developed in this thesis. With the help of the use cases, we have shown the abilities of the developed system and have indirectly proven the viability of the dialogue model. The comparison with similar approaches highlights the differences and strengths of NADIA. While Voice XML focusses on telephony integration and ASR/TTS support, we try to facilitate the development process by making use of dialogue acts, language generation, mixed initiative, natural language understanding modules, automatic subdialogues and a simple method for interfacing external data sources.

The aim of the external evaluation was to get a clearer and independent view on the results of this thesis. Experts from both academia (professors) and industry (project leaders) could prove the demand for such a system and certify its importance. They underline the simplicity of the presented approach and claim that it contributes to a simple creation of more natural dialogues. In a user study, non-expert participants with a basic understanding in speech technologies had to model different dialogues with NADIA and had to evaluate the development process and the results. Major difficulties in creating the dialogues could not be detected. Once the structure of the model had been understood, development was straightforward and the participants stated that this approach is simple and allows the creation of natural dialogues.

In the introduction we have postulated five theses that we want to address now.

T1: *User utterances relate to different levels of the whole system. A differentiation in relation to the scope of an utterance is the basis to understand the utterance and to prevent misunderstandings.*

True. A question is always accompanied by a certain context in that it has to be interpreted. We use *tasks* to structure the dialogue and *selectors* to identify the very task that corresponds to the current user utterance. Without tasks it would not be possible to decide if a request like “Please tell me the city!” relates to the weather information or the trip information system.

T2: *The same intention can be expressed by various formulations. A model that links these variations to the same meaning simplifies processing. However, we still need*

a detailed expressiveness in order to generate language. Hence, the model needs to support both views: interpretation and generation.

True. We use answer types to describe what a system question asks for. This allows us to map formulations like “*When do you want to depart?*” and “*Please tell me your departure date!*” to an utterance that asks for a date. By this means, we can use predefined natural language understanding modules for the interpretation of the user’s answer to that question. In order to automatically generate system questions, we use abstract concepts that describe the intention of the question. This information is specified in the context level of the AQD (e.g. *begin of trip*). Moreover, we use dialogue acts to recognise user utterances as information-seeking, information-providing or action-requesting. This allows us to process an utterance that is formulated as a request as a phrase that seeks for information.

T3: *Questions and answers together form a vital context and should be modelled and processed in one unit.*

True. The strong relationship between question and answer can often be seen in the grammatical and lexical alignment. More importantly, an answer is not interpretable without the context of the question, e.g. the answer “*Three*” can only be correctly understood if we know that the question was “*With how many persons do you travel?*” and not “*For how many days do you want to have a weather forecast?*”. We use ITOs to model both the question and the possible answers in one unit. The AQD – as part of the ITO – defines an answer type to connect to the correct language understanding modules and a context for being able to create a suitable surface form of the question.

T4: *Adaptive system utterances contribute to a higher user acceptance.*

True. In our survey on SDS we have found that most people prefer human-like dialogues. This requires the system not only to understand the user’s aims but also to reflect his style of behaviour. A colloquial user request should not be replied with a very formal answer in order to simulate a believable character. Also, most people want the system to behave politely and start the conversation with an open question instead of uttering an artificial and machine-like prompt. Still, some users prefer a short, cold, and command-like style. Consequently, we consider a *natural* style to be a way of interaction that the user expects and wishes for. This requires an adaptive system which helps to transport a certain personality that people like and accept. However, the most important quality of a dialogue system still is the capability of correctly understanding the user and fulfilling his wishes.

T5: *The development of dialogue systems is still very complex today and needs to be simplified in order to be able to create natural dialogues.*

True. Experience from our own research projects, lab sessions with students and talks with experts have let us to the conclusion that not only specific linguistic

issues need thorough research, but also the modelling and engineering of a whole dialogue system itself is a candidate for intensive research and major improvement. We found that aspects like dialogue act theory and language generation methods have not been integrated in current dialogue development environments. This eliminates the chance to create an adaptive dialogue system which is one of the major flaws when it comes to the development of natural dialogues. Moreover, the structure of the dialogue has been neglected a long time. Many development environments only allow you to specify a string that should be synthesised and a simple grammar for recognising a keyword that does not need to be further interpreted. Often, the developer needs to reimplement these grammars and the understanding module (if any) again for every new project. This also holds for missing back-end accessors and a lack of the declaration of a dialogue strategy. For this reason, we have designed and implemented a model that addresses the demand for a reusable dialogue specification that separates the dialogue engine and logic from the information demand, includes approaches from the latest research and allows to change the dialogue behaviour by parameters. Our expert interviews prove the necessity for such a system and certify that it really simplifies dialogue development and contributes to more natural dialogues.

15.1 Contributions

In this thesis we have addressed different research questions and contributed to a number of aspects on spoken dialogue systems. Some of those issues have already been named in Section 1.3 and shortly commented in Section 1.4. By discussing them now, we also give a short summary of the main contributions.

Q1: *What problems exist in the interaction between man and machine and how do humans prefer to communicate with automated systems?*

In order to improve dialogue systems, we first needed to know what users think of current systems and what they dislike about them. By conducting a survey, we found that only 2% think that there is nothing to improve. Most people complained about the tedious question-answer sequences (63%) with an endless list of choices (52%) and rather want to actively tell the system their concern (80%) instead of waiting for the right question to be asked by the system. Also, 58% would appreciate the understanding of full sentences instead of only keywords. Moreover, we asked the participants to choose their preferred dialogue style from three examples. 71% voted for the most human-like dialogue and also stated that they consider a polite style important.

Q1.1: *What features do current dialogue systems support?*

Many dialogue systems base on a finite-state approach. The dialogue designer specifies a set of questions that is processed in exactly that order. The user

has no possibility to influence the dialogue by giving over-informative answers or by changing the topic. More sophisticated dialogue systems support a “*limited mixed initiative*” (McTear, 2004) which allows the user to give more information in one answer than has been asked for. However, true mixed initiative and a human-like behaviour are only supported by few research prototypes. This also includes the formulation of system prompts which are often very rigid and simple and do not adapt to the user.

Q1.2: *What do users expect from dialogue systems?*

We have already learned that people like to be able to tell the computer their concern instead of answering lots of questions. Furthermore, over-informative answers and a true mixed initiative that supports subdialogues is important for a good usability. Referring to the survey, we can conclude that also human-like and polite formulations are a critical factor. In general, users expect that their request is processed correctly. If they are able to reach their aim with the user interface, the most important hurdle has been overcome. However, once this requirement has been fulfilled, the difficulty in handling and the appearance of the interface becomes important for the rating of the whole system.

Q1.3: *What features make a dialogue natural?*

A dialogue system is natural if it behaves as the user expects and successfully helps him to reach his goals. This may in some cases also be a very restricted dialogue if that is what the user wants (also dialogues between two persons may be somehow restricted and predictable, e.g. an employee at the registration desk of a big event asking everybody for the name). However, in most cases people prefer more verbose sentences with the ability to give several pieces of information at once and by this means having the opportunity to control the dialogue flow. In this connection, they also like open-ended questions that just ask the user what he wishes. Another aspect concerns the formulation of the system utterances. A user and context-dependent formulation contributes to a more natural appearance of the dialogue. As defined in Section 3.2.4, a natural dialogue system should support mixed initiative, be adaptive, make use of implicit confirmation, verify uncertain information, let the user correct given information, interpret over-informative answers, understand negations and colloquial language, resolve anaphora and formulate system utterances in a socially appropriate way.

Based on these questions and findings, we have addressed Bringert’s outcome, that current dialogue systems are not natural enough, not usable enough and not cheap enough. We have identified what needs to be improved in current dialogue systems and what users expect from future systems. Instead of trying to create another dialogue system that is better in one particular aspect and tailored to a specific application, we have aimed at developing a generic dialogue model and developing methodology that addresses many

aspects and that can be used in several projects. We see the reason for so many rudimentary dialogues in the lack of an easy-to-use dialogue development environment. Our model is the basis for such a programme.

Q2: *How can a natural dialogue be described so that it is automatically processable by a dialogue engine?*

First we need to specify a class of dialogues that we want to model. In this thesis we focus on information-seeking and control dialogues rather than question-answering-systems, chatbots or agents. This type of dialogue has a salient structure, as it consists of a greeting, a sequence in which all the necessary information will be retrieved, and finally the presentation of the results. Here, any task that we want to solve by speech consists of a list of attributes that the user needs to provide and a back-end request that the system should perform. This reoccurring structure is the basis for a model that can be automatically processed by a dialogue engine. This model, that aims at facilitating dialogue development by offering a structured development methodology and reusable modules (e.g. language understanding or language generation), will be further described by answering the following research questions.

Q2.1: *How can we classify the intention of a user utterance and its effects on the system independently from the surface form?*

In our model we use dialogue acts to classify user utterances. In contrast to other approaches, we do not classify by the pragmatic type of the utterance (question, wish, request, want, demand, . . .) but instead classify by the effect on the dialogue system, i.e. we analyse if the user's utterance invokes a lookup of information, presents the system with new information or asks for an action to be executed. The intention of a question, a wish or a request might be the seeking for information, e.g. "How is the weather in Edinburgh?" or "Tell me the temperature in Glasgow!". Consequently, these utterances are classified as *information-seeking*. On the other hand, what seems like a question can actually be a request to do something (this is called an *indirect speech act* or the *primary illocution*), e.g. "Could you please switch the light on?". This question is classified as *action-requesting* in our approach. Finally, if the user tells the system "To Glasgow" as an answer to the question "Where do you want to go?", we speak of an *information-providing act*.

Q2.2: *How can the information demand of a dialogue be modelled?*

We describe a dialogue by a list of domain-specific tasks like *weather forecast* or *bus timetable*. Every task consists of a number of attributes that the system needs to collect for being able to execute a query. These attributes relate to questions that ask for them and answers that are valid in the scope of the question. Hence, as we have already observed, questions and answers should

be modelled within one unit. In our model, we call this unit *Information Transfer Object* (ITO). It includes a backup surface form of the question in case language generation fails, a marker if this attribute is compulsory and an *Abstract Question Description* (AQD). The AQD specifies what the question asks for and what are the valid answers (type), what it refers to (context) and how it should be formulated (form). This information is used for both language generation and language understanding. Every ITO is embedded into a task that also contains an *action*. Actions describe the query or request that needs to be executed on an external system. Based on this model we have described the information demand of several dialogues and proven its functionality with the NADIA system.

Q2.3: *How can we set the basis for a system that adapts its style to the user?*

With the help of the AQD we can describe questions by abstract concepts. This is the basis for a language generator to create surface forms that reflect a certain character or that adapts to the user's style. Apart from the linguistic style, also the dialogue behaviour can be adapted to the user by changing the values of the dialogue parameters and thus changing the type of initiative.

Q2.4: *Does the resulting model allow the simple creation of natural dialogues?*

The resulting model has been transferred into an XML-based description which makes it machine-processable. We have also developed a dialogue engine that is able to process the dialogue description and interact with the user. Our final evaluation by both speech dialogue experts and users has proven that the approach is functional, does facilitate the development of dialogue systems in general and also contributes to the creation of more natural dialogues.

We now conclude by summarising the contributions of this thesis. Our aim is to improve state of the art speech dialogues by setting the basis for an easier development of more natural dialogues in the context of speech-based information and control systems. We have analysed the state of the art, identified the user's needs, presented a machine-processable model that addresses these needs and facilitates the development of natural dialogue systems and finally have evaluated the viability of this model by implementing a dialogue engine on that basis.

In the first part of the thesis, different (socio-)linguistic and computational aspects of dialogue systems that build the basis for the subsequent chapters have been regarded. This includes relevant definitions of the terms and methods as well as the description of important models. In particular, we dealt with the differences between communication and interaction, the interpretation of the term *natural language*, different communication models, the architecture and characteristics of natural dialogue systems as well as the classification of utterances by dialogue acts. Here, we build conventions for widely used terms in order to make clear our view and reduce the ambiguity of these terms across different authors and disciplines. These conventions can be used in any dialogue related

work and contribute to dialogue research on a very abstract level. Then we have presented related work in order to place this thesis in the right context and delimit from existing research.

In the second part we have conducted a user study and a survey in order to learn how users interact with speech dialogue systems, what they expect of them and what features make these systems more natural. This was necessary to derive requirements for the dialogue model. In summary, it can be said that users are different and have diverse opinions about the preferred system style, which makes adaptive systems inevitable. Adaptation is also necessary to reflect the user's language or to express a certain character. However, most users prefer a human-like dialogue that starts with an open-ended question, allows over-informative answers and enables the user to take control of the dialogue by using subdialogues. They also favour polite and full sentences instead of machine-like prompts.

Based on these findings, in the third part we have built a model for information-seeking dialogue systems that addresses these issues. As a first step, we analysed the effect of utterances towards the system and related them also to the effect on the back-end, i.e. we distinguished between the type of access of the dialogue system itself and the query that is executed on any external service. This led to system-oriented dialogue acts that describe the goal that a user has when making a request and determine if data needs to be retrieved, updated or stored. This is important in order to not confuse the question “*Could you please switch the light off?*” with the request “*Switch the light off!*”. Because knowing that an utterance is *information-seeking* is not detailed enough for an interpretation, we focussed on a more granular categorisation of questions and their answers. The result is an abstract question description that includes information about the type, context and form of a system question. The type specifies what the question asks for (e.g. a temporal question) whereas the context describes in what situation the question is used (e.g. to ask for the begin of a trip). The form is used for concept-to-text language generation which creates surface forms of system questions in different politeness and formality variations based on the AQD, an OpenCCG grammar and an ontology. These findings have then been combined into a dialogue model that describes the information demand and the search query of an information-seeking dialogue. This model contains *tasks* which group questions that belong to the same goal, *selectors* which activate tasks depending on the user utterance, *ITOs* that represent the question and its possible answers (and connect to NLU modules), and *actions* that execute the desired operation on the back-end. Afterwards, we have developed a dialogue engine (NADIA) that is able to process this model in order to test its operability in the fourth part.

The final evaluation has proven that the model, as well as the engine, is fully functional. Experts have confirmed that the approach is useful, addresses a real demand and contributes to an easier development of more natural dialogues. Although the goals of this thesis have been accomplished, there is still work to be done to create a dialogue system that is indistinguishable from a human being. The NADIA system is a good basis for further research, which we will briefly outline in the next section.

15.2 Limitations and Future Work

The development of dialogue systems is a very complex area of research that includes many different branches of science like computer science, linguistics, sociology and psychology. Even within natural language processing, many subjects – like natural language understanding, natural language generation, automatic speech recognition, text synthesis or dialogue management – are covered and need to be integrated. Hence, the modelling of such a system and its improvement is a complex task as well. Naturally, not all aspects or issues can be addressed and solved in a single thesis. However, we have already seen, that our model is valid and functional and that the resulting NADIA system facilitates the development of natural dialogues. Nevertheless, there is still work to be done, that we address now and that may be the basis for future theses.

Two aspects for improvement, which are currently under development, have already been mentioned: A more sophisticated language understanding and an easy to use development environment. Moreover, anaphora resolution and ellipsis detection would further improve the NLU component, as the following dialogues demonstrate.

Dialogue 15.57:

- S: *How may I help you?*
U: *How is the weather in Aberdeen today?*
S: *It's rainy at 12°C.*
U: *And in Edinburgh?*
S: *Cloudy at 14°C.*
U: *What about the next days?*

Dialogue 15.58:

- U: *Is the lamp in the basement switched on?*
S: *Yes!*
U: *Please switch it off then!*
S: *Okay, the light has been switched off.*

On the practical side, language generation grammars can be extended with new sentence structures and the lexicon could be enlarged with more words. The extension of the grammar would make possible the generation of explicit verification questions like “*So you want to go to Edinburgh, right?*” or implicit verification questions like “*When do you want to go to Edinburgh?*”. Also, we can think of using the AQD, and thus language generation, not only for system questions but also for answers. For this purpose, the *actions* would need to be extended with the AQD.

Another field of improvement is conditional ITOs which can be used to implement optional questions that depend on the previous answer. At the moment, the model only supports hidden questions that are used to process answers that the user gives without being asked for or to allow to further restrict the search query if there are too many results. However, conditional ITOs are useful to ask a question only if certain circumstances apply, e.g. the question for the credit card number needs only to be asked if the customer selected the credit card as payment method. Otherwise the system could

ask for the bank account details.

Also the grouping of ITOs or the creation of sub-ITOs is an interesting way to further improve the system. This allows the arrangement of ITOs by their topic or semantic proximity. In a travel agency task, the questions for the start and end date are semantically closer to each other than the questions for the start date and the destination. These semantically close questions are also likely to be answered in one turn by being over-informative. Hence, instead of asking them separately, we could pose a more general question like *“When do you want to travel?”* and only in case of problems switch to the more specific questions *“When do you want to start?”* and *“When do you want to return?”*. Here, language generation can also help to generate questions in different levels of detail. In this connection shared frames, i.e. the passing of frames between tasks, get interesting as well. Imagine your dialogue consists of two tasks, one for getting trip information and one for booking a trip. In this case it is very useful to pass the collected information from the first to the second task to provide the user with a seamless interaction.

We can easily see that there is still a lot of work to do, although we have only named some of the various aspects of dialogue systems research. We believe that the developed model and dialogue engine is a good basis for further research and a useful tool to create complex dialogues within university courses.

A Appendix

A.1 Maximum Entropy Classifier Data

Training set:

- information-seeking
 - How is the weather in Paris
 - Tell me the weather in Paris
 - Where do you want to go
 - What is your destination
 - Please tell me where you want to go
 - Please tell me your destination
- action-requesting
 - Could you please turn the light on
 - Turn the light on
 - Please switch the light off
 - Can you switch the light off
 - Could you turn off the computer
- information-providing
 - To London
 - I want to go to London
 - Paris
 - I'd like to go to Paris

Test set with results in the form *with verb cues* / *without verb cues*:

- Could you recommend a hotel for next week → seek / action
- Close the window → action / unknown (seek)

- When do you want to come back → seek / seek
- Can you close the door → action / action
- Paris is nice → prov / action
- Two adults please → prov / unknown (seek)
- I want to have a non smokers room → prov / unknown (seek)
- How much would that be → seek / seek
- Can I have a double room → prov / action
- Three persons → prov / prov
- London would be great → prov / unknown (seek)
- I really prefer London → prov / unknown (seek)
- I want to know about hotels in London → seek / unknown (seek)

A.2 Politeness Scores for Different Question Types

Table A.1: Politeness scores for fact questions

Question Type	Please	Subjunctive	Politeness Score (-2..5)
N	false	false	-2
NRequest	false	false	-1
WhRequest	false	false	-1
N	true	false	0
NRequest	true	false	1
WhRequest	true	false	1
WhQuestion	false	false	2
CanWhQuestion	false	false	3
CanNQuestion	false	false	3
CanWhQuestion	true	false	4
CanNQuestion	true	false	4
CanWhQuestion	false	true	4
CanNQuestion	false	true	4
CanWhQuestion	true	true	5
CanNQuestion	true	true	5

Table A.2: Politeness scores for decision questions

Question Type	Please	Subjunctive	Politeness Score (-2..5)
N	false	false	-2
N	true	false	-1
YNQuestion	false	false	0
YNQuestion	false	false	1
YNQuestion	false	false	2
YNQuestion	false	false	3
YNQuestion	false	false	4
YNQuestion	false	false	5

A.3 Dialogue Definition

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet href="/nadia/dialog.xsl" type="text/xsl"?>
3 <n:dialog xsi:schemaLocation="http://mmberg.net/nadia_schema1.xsd" xmlns:n="
  http://mmberg.net/nadia" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" name="dummy2">
4   <start_task_name>start</start_task_name>
5   <global_language>en</global_language>
6   <global_politeness>2</global_politeness>
7   <global_formality>2</global_formality>
8   <useSODA>true</useSODA>
9   <allowSwitchTasks>true</allowSwitchTasks>
10  <allowOverAnswering>true</allowOverAnswering>
11  <allowDifferentQuestion>true</allowDifferentQuestion>
12  <allowCorrection>true</allowCorrection>
13  <tasks>
14    <task name="start">
15      <selector>
16        <bagOfWordsTaskSelector>
17          <word>hello</word>
18        </bagOfWordsTaskSelector>
19      </selector>
20      <itos>
21        <ito name="welcome">
22          <AQD>
23            <type>
24              <answerType>open_ended</answerType>
25            </type>
26          </AQD>
27          <fallback_question>How may I help you?</fallback_question>
28          <group>0</group>
29          <index>0</index>
30          <required>false</required>
31          <useLG>false</useLG>
32        </ito>
33      </itos>
34    </task>
35    <task name="getTripInformation">
36      <selector>

```

Appendix

```
37     <bagOfWordsTaskSelector>
38         <word>travel</word>
39         <word>book</word>
40         <word>journey</word>
41         <word>trip</word>
42     </bagOfWordsTaskSelector>
43 </selector>
44 <itos>
45     <ito name="getDepartureCity">
46         <AQD>
47             <context>
48                 <reference>trip</reference>
49                 <specification>begin</specification>
50             </context>
51             <form>
52                 <temporalOpener>>false</temporalOpener>
53             </form>
54             <type>
55                 <answerType>fact.named_entity.non_animated.location.city</
                    answerType>
56             </type>
57         </AQD>
58         <fallback_question>Where do you want to start?</fallback_question
                    >
59         <group>0</group>
60         <index>0</index>
61         <required>>false</required>
62         <useLG>>true</useLG>
63     </ito>
64     <ito name="getDestinationCity">
65         <AQD>
66             <context>
67                 <reference>trip</reference>
68                 <specification>end</specification>
69             </context>
70             <form>
71                 <temporalOpener>>false</temporalOpener>
72             </form>
73             <type>
74                 <answerType>fact.named_entity.non_animated.location.city</
                    answerType>
75             </type>
76         </AQD>
77         <fallback_question>Where do you want to go?</fallback_question>
78         <group>0</group>
79         <index>0</index>
80         <required>>false</required>
81         <useLG>>true</useLG>
82     </ito>
83     <ito name="getNumberOfPersons">
84         <AQD>
85             <type>
86                 <answerType>fact.quantity</answerType>
87             </type>
88         </AQD>
89         <fallback_question>For how many persons?</fallback_question>
90         <group>0</group>
91         <index>0</index>
92         <required>>false</required>
93         <useLG>>false</useLG>
94     </ito>
95     <ito name="getDate">
```

Appendix

```
96         <AQD>
97             <context>
98                 <reference>trip</reference>
99                 <specification>begin</specification>
100             </context>
101             <form>
102                 <temporalOpener>>false</temporalOpener>
103             </form>
104             <type>
105                 <answerType>fact . temporal . date</answerType>
106             </type>
107         </AQD>
108         <fallback_question>When do you want to leave?</fallback_question>
109         <group>0</group>
110         <index>0</index>
111         <required>false</required>
112         <useLG>true</useLG>
113     </ito>
114     <ito name="getDate">
115         <AQD>
116             <context>
117                 <reference>trip</reference>
118                 <specification>end</specification>
119             </context>
120             <form>
121                 <temporalOpener>>false</temporalOpener>
122             </form>
123             <type>
124                 <answerType>fact . temporal . date</answerType>
125             </type>
126         </AQD>
127         <fallback_question>When do you want to come back?</
128             fallback_question>
129         <group>0</group>
130         <index>0</index>
131         <required>false</required>
132         <useLG>true</useLG>
133     </itos>
134     <action>
135         <groovyAction>
136             <resultMappings>
137                 <resultMapping>
138                     <message>#price Euro for %getDestinationCity is cheap!</
139                         message>
140                     <resultValue>257</resultValue>
141                     <resultVarName>price</resultVarName>
142                 </resultMapping>
143                 <resultMapping>
144                     <message>#price Euro for %getDestinationCity is expensive!<
145                         /message>
146                     <resultValue>1000</resultValue>
147                     <resultVarName>price</resultVarName>
148                 </resultMapping>
149             </resultMappings>
150             <returnAnswer>true</returnAnswer>
151             <utteranceTemplate>This trip from %getDepartureCity to %
152                 getDestinationCity costs #price Euros.</utteranceTemplate>
153             <code><![CDATA[executionResults.put("price", "257")]></code>
154         </groovyAction>
155     </action>
156 </task>
```


Appendix

```
154 <task name="getWeatherInformation">
155   <selector>
156     <bagOfWordsTaskSelector>
157       <word>weather</word>
158       <word>forecast</word>
159       <word>temperature</word>
160     </bagOfWordsTaskSelector>
161   </selector>
162   <itos>
163     <ito name="getWeatherCity">
164       <AQD>
165         <type>
166           <answerType>fact.named_entity.non_animated.location.city</
167             answerType>
168         </type>
169       </AQD>
170       <fallback_question>For which city do you want to know the weather
171         ?</fallback_question>
172       <group>0</group>
173       <index>0</index>
174       <required>>false</required>
175       <useLG>>false</useLG>
176     </ito>
177   </itos>
178   <action>
179     <httpAction>
180       <returnAnswer>>true</returnAnswer>
181       <utteranceTemplate>The temperature in %getWeatherCity is #result
182         degrees.</utteranceTemplate>
183       <method>get</method>
184       <params>q=%getWeatherCity&mode=xml&units=metric</params>
185       <url>http://api.openweathermap.org/data/2.5/weather</url>
186       <xpath>/current/temperature/@value</xpath>
187     </httpAction>
188   </action>
189 </task>
190 <task name="getWikipediaCityInfo">
191   <selector>
192     <bagOfWordsTaskSelector>
193       <word>wikipedia</word>
194       <word>tell me about</word>
195       <word>know about</word>
196     </bagOfWordsTaskSelector>
197   </selector>
198   <itos>
199     <ito name="getWikiCity">
200       <AQD>
201         <type>
202           <answerType>fact.named_entity.non_animated.location.city</
203             answerType>
204         </type>
205       </AQD>
206       <fallback_question>What city do you want to know more about?</
207         fallback_question>
208       <group>0</group>
209       <index>0</index>
210       <required>>false</required>
211       <useLG>>false</useLG>
212     </ito>
213   </itos>
214   <action>
215     <httpAction>
```

Appendix

```
211     <returnAnswer>true</returnAnswer>
212     <utteranceTemplate>#result</utteranceTemplate>
213     <method>get</method>
214     <params>format=xml& ; action=query& ; prop=extracts& ;
        explaintext& ; exsentences=3& ; titles=%get WikiCity</params
        >
215     <url>http://en.wikipedia.org/w/api.php</url>
216     <xpath>//extract</xpath>
217 </httpAction>
218 </action>
219 <followup>
220   <ito name="anotherOne">
221     <AQD>
222       <type>
223         <answerType>decision</answerType>
224       </type>
225     </AQD>
226     <fallback_question>Do you want to know about other cities?</
        fallback_question>
227     <group>0</group>
228     <index>0</index>
229     <required>>false</required>
230     <useLG>>false</useLG>
231   </ito>
232   <answerMapping>
233     <item key="YES">getWikipediaCityInfo</item>
234   </answerMapping>
235 </followup>
236 </task>
237 <task name="setLightbulb">
238   <act>action</act>
239   <selector>
240     <bagOfWordsTaskSelector>
241       <word>bulb</word>
242       <word>switch</word>
243     </bagOfWordsTaskSelector>
244   </selector>
245   <itos>
246     <ito name="getLightAction">
247       <AQD>
248         <type>
249           <answerType>onoff</answerType>
250         </type>
251       </AQD>
252       <fallback_question>Do you want to switch it on or off?</
        fallback_question>
253       <group>0</group>
254       <index>0</index>
255       <required>>false</required>
256       <useLG>>false</useLG>
257     </ito>
258   </itos>
259   <action>
260     <httpAction>
261       <returnAnswer>true</returnAnswer>
262       <utteranceTemplate>#result</utteranceTemplate>
263       <method>post</method>
264       <params>state=%getLightAction</params>
265       <url>http://mmt.et.hs-wismar.de:8080/Lightbulb/Lightbulb</url>
266       <xpath>//message</xpath>
267     </httpAction>
268   </action>
```

Appendix

```
269 </task>
270 <task name="getLightbulb">
271   <act>seek</act>
272   <selector>
273     <bagOfWordsTaskSelector>
274       <word>bulb</word>
275       <word>switch</word>
276     </bagOfWordsTaskSelector>
277   </selector>
278   <itos/>
279   <action>
280     <httpAction>
281       <returnAnswer>>true</returnAnswer>
282       <utteranceTemplate>#result</utteranceTemplate>
283       <method>get</method>
284       <params>getState</params>
285       <url>http://mmt.et.hs-wismar.de:8080/Lightbulb/Lightbulb</url>
286       <xpath>//message</xpath>
287     </httpAction>
288   </action>
289 </task>
290 <task name="guessNumber">
291   <act>seek</act>
292   <selector>
293     <bagOfWordsTaskSelector>
294       <word>guess</word>
295       <word>number</word>
296       <word>play</word>
297     </bagOfWordsTaskSelector>
298   </selector>
299   <itos>
300     <ito name="getNumber">
301       <AQD>
302         <type>
303           <answerType>fact.quantity</answerType>
304         </type>
305       </AQD>
306       <fallback_question>Guess a number between 1 and 99!</
307         fallback_question>
308       <group>0</group>
309       <index>0</index>
310       <required>>false</required>
311       <useLG>>false</useLG>
312     </ito>
313   </itos>
314   <action>
315     <httpAction>
316       <resultMappings>
317         <resultMapping>
318           <message>was too big.</message>
319           <redirectToTask>guessNumber</redirectToTask>
320           <resultValue>TOO_BIG</resultValue>
321           <resultVarName>result</resultVarName>
322         </resultMapping>
323         <resultMapping>
324           <message>was too small.</message>
325           <redirectToTask>guessNumber</redirectToTask>
326           <resultValue>TOO_SMALL</resultValue>
327           <resultVarName>result</resultVarName>
328         </resultMapping>
329       </resultMappings>
330     </httpAction>
331   </action>
332   <message>it is! Congratulations!</message>
```

```

330         <resultValue>CORRECT</resultValue>
331         <resultVarName>result</resultVarName>
332     </resultMapping>
333 </resultMappings>
334 <returnAnswer>true</returnAnswer>
335 <utteranceTemplate>%getNumber </utteranceTemplate>
336 <method>get</method>
337 <params>guess=%getNumber</params>
338 <url>http://mmt.et.hs-wismar.de:8080/NumberGuessing/
    NumberGuessing</url>
339 <xpath>//code</xpath>
340 </httpAction>
341 </action>
342 <followup>
343     <ito name="playAgain">
344         <AQD>
345             <type>
346                 <answerType>decision</answerType>
347             </type>
348         </AQD>
349         <fallback_question>Do you want to play again?</fallback_question>
350         <group>0</group>
351         <index>0</index>
352         <required>>false</required>
353         <useLG>>false</useLG>
354     </ito>
355     <answerMapping>
356         <item key="YES">guessNumber</item>
357     </answerMapping>
358 </followup>
359 </task>
360 </tasks>
361 </n:dialog>

```

A.4 Dialogue Schema

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    attributeFormDefault="unqualified" elementFormDefault="qualified"
    targetNamespace="http://github.com/mmberg/nadia" xmlns:xs="http://www.w3.org
    /2001/XMLSchema">
3 <xs:element name="dialog">
4 <xs:complexType>
5 <xs:sequence>
6 <xs:element name="tasks">
7 <xs:complexType>
8 <xs:sequence>
9 <xs:element name="task" maxOccurs="unbounded">
10 <xs:complexType>
11 <xs:sequence>
12 <xs:element name="action" minOccurs="0">
13 <xs:complexType>
14 <xs:sequence>
15 <xs:choice>
16 <xs:sequence>
17 <xs:element name="path" type="xs:string" />
18 <xs:element name="className" type="xs:string" />
19 </xs:sequence>
20 <xs:sequence>
21 <xs:element name="code" type="xs:string" />

```

```

22     </xs:sequence>
23   </xs:choice>
24   <xs:sequence>
25     <xs:element name="utteranceTemplate" type="xs:string" />
26     <xs:element name="returnAnswer" type="xs:boolean" minOccurs="0" />
27   </xs:sequence>
28 </xs:sequence>
29 </xs:complexType>
30 </xs:element>
31 <xs:element name="itos">
32   <xs:complexType>
33     <xs:sequence>
34       <xs:element name="ito" maxOccurs="unbounded">
35         <xs:complexType>
36           <xs:sequence>
37             <xs:element name="AQD">
38               <xs:complexType>
39                 <xs:sequence>
40                   <xs:element name="type" minOccurs="0">
41                     <xs:complexType>
42                       <xs:sequence>
43                         <xs:element name="answerType" type="xs:string" />
44                       </xs:sequence>
45                     </xs:complexType>
46                   </xs:element>
47                 </xs:sequence>
48               </xs:complexType>
49             </xs:element>
50             <xs:element name="fallback_question" type="xs:string" />
51             <xs:element name="group" type="xs:unsignedByte" minOccurs="0" />
52             <xs:element name="index" type="xs:unsignedByte" minOccurs="0" />
53             <xs:element name="required" type="xs:boolean" minOccurs="0" />
54             <xs:element name="useLG" type="xs:boolean" minOccurs="0" />
55           </xs:sequence>
56           <xs:attribute name="name" type="xs:string" use="optional" />
57         </xs:complexType>
58       </xs:element>
59     </xs:sequence>
60   </xs:complexType>
61 </xs:element>
62 <xs:element name="selector">
63   <xs:complexType>
64     <xs:sequence>
65       <xs:element name="bagOfWords">
66         <xs:complexType>
67           <xs:sequence>
68             <xs:element name="word" type="xs:string" maxOccurs="unbounded" /
69             >
69           </xs:sequence>
70         </xs:complexType>
71       </xs:element>
72     </xs:sequence>
73   </xs:complexType>
74 </xs:element>
75 </xs:sequence>
76 <xs:attribute name="name" type="xs:string" use="optional" />
77 </xs:complexType>
78 </xs:element>
79 </xs:sequence>
80 </xs:complexType>
81 </xs:element>
82 </xs:sequence>

```

Appendix

```
83 <xs:attribute name="name" type="xs:string" use="optional" />
84 </xs:complexType>
85 </xs:element>
86 </xs:schema>
```

A.5 NADIA Manual

Nadia stands for *Natural Dialogue System* and is a set of components that deals with the creation of spoken dialogue systems. While common standards like *VoiceXML* are widely used in industry, it is still quite a challenge to design dialogues that make use of well established theories from research projects. Although it is easy to create simple dialogues, we cannot specify the dialogue strategy, we don't have an integrated support for language generation and we don't have parsers for common question types at hand. Moreover, we need a complex IVR architecture which means a lot of effort to setup a dialogue development environment on restricted university computers. Alternatively, it is possible to create a dialogue system manually based on speech recognition/synthesis APIs like the Microsoft SAPI. Again, there is no support regarding the specification of the dialogue and its behaviour and you would have to create basic modules for every project again and again.

Hence, the goal of NADIA is to offer an approach to design dialogues without having to code NLU modules/parsers, dialogue strategies or complex behaviour. The basis is a dialogue model that can be automatically processed by NADIA. The task of the dialogue designer is only to specify the questions and their properties. You will learn more about this model later. Currently, the main usage of NADIA is in dialogue system related university courses.

A.5.1 Run Modes

Nadia comes as a jar-file that can be run locally. We offer two run modes that can be specified by command line arguments:

- console
- embedded REST service (Jetty)

Moreover, we offer a war-file that can be deployed on a Jetty or Tomcat server.

Command Line Arguments

```
usage: nadia [-f ] [-h] [-i ] [-r ] [-s ]  
  
-f,--file => specify dialogue path and file , e.g. -f /path/to/dialogue1.xml  
The root of the path refers to the root of the war-file or the directory  
where the jar resides in  
  
-h,--help => print this message  
  
-i,--interface => select user interface , e.g. -i rest or -i console  
  
-r,--resource => load dialogue (by name) from resources , e.g. -r dialogue1  
No path required. Indicate a file name (without extension) that exists in  
the dialogue resources directory /res/dialogues.
```

```
-s,--store => load dialogue (by name) from internal store , e.g. -s
dialogue1
Loads an in-memory dialogue , i.e. a dialogue that has been designed not as
XML but as a Java Object within Nadia and given a name.
```

REST

If you have started NADIA as a REST service (locally or on a Java Application Server), you can test the application by calling the URL `http(s)://SERVER:PORT/nadia`, e.g. `https://localhost:8080/nadia`. Please note that the embedded server uses `https` while the deployed version only uses `http` (if you don't configure it otherwise).

Communication

As a first step, any client needs to create a new dialogue instance on the server. This is done by a call to `/nadia/engine/dialog` (we always assume the base URL `http(s)://SERVER:PORT`, e.g. `https://localhost:8080` which would result in `https://localhost:8080/nadia/engine/dialog`). The system will return a new URL (e.g. `/nadia/engine/dialog/d1`) as a `Location` response header that the client must use for every subsequent communication. This url refers to the created dialogue and adds the dialogue ID, e.g. `d1`. Moreover, the first system utterance is sent to the client in the parameter `systemUtterance`. The first mentioned URL loads the default dialogue. Alternatively you could call `/nadia/engine/dialog/load` and pass the content of a Nadia XML Dialogue Description as a `dialogxml` form parameter (file upload).

In the next step, the user wants to answer the question. You will need to get the text from the user and send it to the received URL in the parameter `userUtterance`. Once a dialogue has been created, you can get more detailed information on the loaded dialogue by calling `/nadia/engine/dialog/DIALOG_ID/xml`, e.g. `/nadia/engine/dialog/d1/xml`, and about the status of the dialogue by calling `/nadia/engine/dialog/DIALOG_ID/context`. Moreover, you can get information about the active sessions under `/nadia/engine/status`.

cURL

If you want to test the communication process without programming your own client or using the existing web-interface, you may use `curl` like so:

1. First start the server with the embedded REST interface: `java -jar nadia.jar -i rest`
 - Alternatively you may start the REST service by deploying the war-file to your application server
2. Initialise the dialogue with `curl -k -i -X POST https://localhost:8080/nadia/engine/dialog`

- the parameter `k` disables SSL verification and may be necessary in case of an unsigned SSL certificate
 - the parameter `i` displays not only the content but also the header. Remember that we are interested in the location.
 - please note that we send a POST request without any data. This can also be done with `curl -k -i --data "" https://localhost:8080/nadia/engine/dialog`.
 - you will receive a system utterance (i.e. the first system question), e.g. "How may I help you?"
3. Send your answer, i.e. the user utterance, to the new URL that you received in the location parameter in the last step, e.g. `curl -k --data "userUtterance=I+want+to+book+a+trip" https://localhost:8080/nadia/engine/dialog/d1`
 - please note that the data should be URL-encoded (separate parameters by `&` and replace spaces by `+`)
 - you should see the next system question, e.g. "Where do you want to start?"
 4. Answer the next question as in 3. `curl -k --data "userUtterance=I+want+to+start+in+Edinburgh" https://localhost:8080/nadia/engine/dialog/d1`
 5. Have a look at the current dialogue context: `https://localhost:8080/nadia/engine/dialog/d1/context`
 6. Repeat step 4 until the dialogue is finished or you want to stop.
 - please note that your dialogue may have been destroyed if it had not been accessed in a while
 - to see all the current sessions call `https://localhost:8080/nadia/engine/status`.

A.5.2 Creating the Dialogue Model

The basis for NADIA is a dialogue model. A dialogue designer needs to specify the dialogue description that the NADIA system can process. There are two approaches to create such descriptions:

- XML
- Java

In order to create the model in Java you should use the *nadia-model project* that you also find on Github (soon). The definition in XML is a bit more effortful and error-prone. That's why we are creating a graphical development environment that helps you to create the XML file without having to write XML yourself.

When you specify the model in Java, you can use the `save()` or `saveAs()` methods to generate an XML representation that you can load the NADIA system with.

Dialogues, Tasks and ITOs

Every dialogue consists of tasks, i.e. a group of questions that ask for all the information that are needed to execute this very task. A task could be to get the weather or to get trip information. Every XML file defines exactly one dialogue. However, to support different functions, every dialogue can consist of several tasks. Tasks are composed of ITOs. An ITO is an *information transfer object* and represents the system question and the possible user answers to that question. A very simple dialogue can be created like this:

```
Dialog dialog = new Dialog("helloworld");

//a dialog consists of tasks
Task task1=new Task("getCities");
dialog.addTask(task1);

//a task consists of ITOs (Information Transfer Objects)
ITO ito;
AQD aqd;

//1
ito=new ITO("getDepartureCity", "Where do you want to start?", false);
task1.addITO(ito);
//an ITO is associated with AQDs (Abstract Question Description)
aqd=new AQD();
aqd.setType(new AQDType("fact.named_entity.non_animated.location.city"));
ito.setAQD(aqd);

//2
ito=new ITO("getDestinationCity", "Where do you want to go?", false);
task1.addITO(ito);
//an ITO is associated with AQDs
aqd=new AQD();
aqd.setType(new AQDType("fact.named_entity.non_animated.location.city"));
ito.setAQD(aqd);
```

This will generate a dialogue (`helloworld`) that consists of one task (`getCities`). This task consists of two questions: one that asks for the departure city and one that asks for the destination city (assuming that we are going to build a travel information system). If you save this, the XML will look something like this (this is a simplified extract):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dialog name="helloworld">
<tasks>
  <task name="getCities">
    <itos>
      <ito name="getDepartureCity">
        <AQD>
          <type>
            <answerType>fact.named_entity.non_animated.
              location.city</answerType>
          </type>
        </AQD>
```

```
<fallback_question>Where do you want to start?</  
  fallback_question>  
<useLG>>false</useLG>  
</ito>  
<ito name="getDestinationCity">  
  <AQD>  
    <type>  
      <answerType>fact.named_entity.non_animated.  
        location.city</answerType>  
    </type>  
  </AQD>  
<fallback_question>Where do you want to go?</  
  fallback_question>  
<useLG>>false</useLG>  
</ito>  
</itos>  
</task>  
</tasks>  
</dialog>
```

If you start NADIA and do not specify a dialogue file, the default dialogue will be loaded. So if you want to use the dialogue that you have just created, you need to run Nadia with `-f path/to/file/name.xml` or call `https://localhost:8080/nadia/index2.html` (here you can upload your own dialogue). Afterwards (if you have initialised your dialogue) you can take a look at a box visualisation of your dialogue under `https://localhost:8080/nadia/engine/dialog/d1/xml`.

So what can we do now with this dialogue? Actually, not much more than answering two questions. We don't have specified actions that should be executed. However, we also did not specify the answer. Wait, we did! We specified an answer type `fact.named_entity.non_animated.location.city`. This lets the system know that we ask for cities. We do not need to manually specify a parser. The system already comes with interpretation facilities. However, at the moment and for this demo you can only use the following cities (more sophisticated modules are under development right now):

- Edinburgh
- Glasgow
- Aberdeen
- Portree
- Uig
- Balloch
- Inverness
- Malaig

At this point we also did not make use of language generation and instead used the fallback question. We will come back to language generation at a later point.

Selectors

If you have more than one task, the system needs to decide which task to use. This decision can be made with the help of task selectors. At the moment this is a simple list of keywords (bag of words). After you have created a task,

```
Task task1=new Task("getWeatherInformation");
```

you can add a bag of words:

```
bagOfWords = new ArrayList<String>(Arrays.asList("weather", "forecast",  
"temperature"));  
task1.setSelector(new BagOfWordsTaskSelector(bagOfWords));
```

This will result in the following XML code:

```
<selector>  
  <bagOfWordsTaskSelector>  
    <word>weather</word>  
    <word>forecast</word>  
    <word>temperature</word>  
  </bagOfWordsTaskSelector>  
</selector>
```

Actions

A dialogue system needs to do something of course, i.e. we want to carry out a certain action. Often, this is the execution of a parametrised query to get information like the weather in a specific city. To execute actions, we use action objects. At the moment we have three different types:

- Java Class Loader (JavaAction): load and execute a compiled Java class
- Groovy (GroovyAction): write a Groovy Script
- HTTP (HTTPAction): send a POST or GET request, get an XML answer and extract the result via XPath

To get the weather from *openweathermap.org*, you need to access the following URL: <http://api.openweathermap.org/data/2.5/weather?q=London&mode=xml>. We can realise that by using an HTTP Action:

```
HTTPAction httpaction=new HTTPAction("#result");  
httpaction.setUrl("http://api.openweathermap.org/data/2.5/weather");  
httpaction.setMethod("get");  
httpaction.setParams("q=London&mode=xml&units=metric");  
httpaction.setXpath("/current/temperature/@value");  
task1.setAction(httpaction); //assuming task1 has been created before
```

Of course we do not want to always get the weather in London. That's why we use the name of the city that the user has mentioned before:

```
httpaction.setParams("q=%getWeatherCity&mode=xml&units=metric");
```

Appendix

The name after the percentage sign refers to the ITO where the user has been asked for the city name, i.e. percentage signs introduce references to ITOs and sharps refer to the name of the result variable. For *HTTPActions* this is always `result`. Moreover, we don't want the system to just say the temperature. Instead we expect a full sentence. That's why we need to change the `utteranceTemplate` in the constructor:

```
HTTPAction httpaction=new HTTPAction("The temperature in %getWeatherCity
is #result degrees.");
```

If we combine this with an ITO, the resulting XML may look like this:

```
<task name="getWeatherInformation">
  <selector>
    <bagOfWordsTaskSelector>
      <word>weather</word>
      <word>forecast</word>
      <word>temperature</word>
    </bagOfWordsTaskSelector>
  </selector>
  <itos>
    <ito name="getWeatherCity">
      <AQD>
        <type>
          <answerType>fact.named_entity.non_animated.location.city<
            /answerType>
        </type>
      </AQD>
      <fallback_question>For which city do you want to know the
        weather?</fallback_question>
      <useLG>>false</useLG>
    </ito>
  </itos>
  <action>
    <httpAction>
      <returnAnswer>>true</returnAnswer>
      <utteranceTemplate>The temperature in %getWeatherCity is #
        result degrees.</utteranceTemplate>
      <method>get</method>
      <params>q=%getWeatherCity&mode=xml&units=metric</params
        >
      <url>http://api.openweathermap.org/data/2.5/weather</url>
      <xpath>/current/temperature/@value</xpath>
    </httpAction>
  </action>
</task>
```

Language Generation

Language Generation is used to facilitate the creation of adaptive and multilingual dialogue systems. In current systems the developer has to specify the exact phrasing of every system question. This has some severe drawbacks:

Appendix

- If the system should also be available in another language, he would need to translate all the prompts
- If the system should adapt to the user’s style of interaction, the developer would need to specify various formulations

This is very time-consuming and inflexible. That’s why we specify system questions by abstract concepts and use language generation methods to create different formulations. Instead of writing “*When do you want to leave?*” we would instruct the computer to generate a question for the departure date in a semi-formal and semi-polite way. This can be done by specifying an answer type (we ask for a *date*) and a context (more information about this date), i.e. the *begin* of a *trip* (which is a departure date). This can be realised with the following representation:

```
<AQD>
  <type>
    <answerType>fact . temporal . date</answerType>
  </type>
  <context>
    <aspect>begin</aspect>
    <reference>trip</reference>
  </context>
  <form>
    <politeness>2</politeness>
    <formality>2</formality>
  </form>
</AQD>
```

If you want to change the formulation without changing the meaning, you only need to change the form parameters. This could result in “*Tell me your departure date*” or “*Could you please specify when you want to start your trip?*”.

A.5.3 Specifying the Dialogue Model with Java

In order to create a dialogue model with Java, you just need to create a new Java project and include the following libraries in your classpath:

- `nadia-model.jar`
- `org.eclipse.persistence.core_2.4.1.v20121003-ad44345.jar`
- `org.eclipse.persistence.moxy_2.4.1.v20121003-ad44345.jar`

Then create a new package and a new class and begin to write your code:

```
package net.mmberg.nadia.model;

public class Modeller {

    public Modeller(){
    }
}
```

```
public static void main(String [] args){
    new Modeller().createDialog();
}

private void createDialog(){

    Dialog d = new Dialog("myDialogue");

    task = new Task("getTemperature");
    task.setSelector(new BagOfWordsTaskSelector(new ArrayList<String>(
        Arrays.asList("temperature","weather"))));
    ito = new ITO("getCity","Please tell me the city.",false);
    aqd = new AQD(new AQDType("fact.named_entity.non_animated.location.
        city"),new AQDContext(), new AQDForm());
    ito.setAQD(aqd);
    task.addITO(ito);
    HTTPAction action = new HTTPAction("The temperature in %getCity is #
        result degrees.");
    action.setMethod("GET");
    action.setUrl("http://api.openweathermap.org/data/2.5/weather");
    action.setParams("q=%getCity&mode=xml&units=metric");
    action.setXpath("/current/temperature/@value");
    task.setAction(action);
    d.addTask(task);

    d.saveAs("", "myDialogue.xml");
}
}
```

In the last step you have to save the dialogue to a file.

A.6 Calculator Example

The following script is a shortened version of <http://cafe.bevocal.com/docs/tutorial/examples.html> and offers a simple Voice XML dialogue that enables the user to perform basic mathematical operations.

```
<form>
  <field name="op">
    <prompt>
      Choose add, subtract, multiply, or divide.
    </prompt>
    <grammar type="application/x-nuance-gsl">
      [add subtract multiply divide]
    </grammar>
    <filled>
      <prompt>
        Okay, let's <value expr="op"/> two numbers.
      </prompt>
    </filled>
  </field>
```

```

<field name="a" type="number">
  <prompt>
    Whats the first number?
  </prompt>
  <filled>
    <prompt> <value expr="a"/> </prompt>
  </filled>
</field>

<var name="result"/>

<field name="b" type="number">
  <prompt> And the second number? </prompt>
  <filled>
    <prompt> <value expr="b"/> Okay. </prompt>

    <if cond="op=='add'">
      <assign name="result" expr="Number(a) + Number(b)"/>
      <prompt>
        <value expr="a"/> plus <value expr="b"/>
        equals <value expr="result"/>
      </prompt>
    <elseif cond="op=='subtract'">
      <assign name="result" expr="a - b"/>
      <prompt>
        <value expr="a"/> minus <value expr="b"/>
        equals <value expr="result"/>
      </prompt>
    <elseif cond="op=='multiply'">
      <assign name="result" expr="a * b"/>
      <prompt>
        <value expr="a"/> times <value expr="b"/>
        equals <value expr="result"/>
      </prompt>
    <else/>
      <assign name="result" expr="a / b"/>
      <prompt>
        <value expr="a"/> divided by <value expr="b"/>
        equals <value expr="result"/>
      </prompt>
    </if>

    <clear/>
  </filled>
</field>
</form>

```

Listing A.1: Voice XML Calculator

We now implement a similar dialogue with NADIA. Please note that we use inline Groovy scripting instead of an external back-end in order to get closer to the functionality of the Voice XML script.


```

<task name="calculator">
  <selector>
    <bagOfWordsTaskSelector>
      <word>math</word>
      <word>calculator</word>
    </bagOfWordsTaskSelector>
  </selector>
  <itos>
    <ito name="number1">
      <AQD>
        <type>
          <answerType>fact.quantity</answerType>
        </type>
      </AQD>
      <fallback_question>Please tell me the first number!</
        fallback_question>
      <required>true</required>
      <useLG>>false</useLG>
    </ito>
    <ito name="number2">
      <AQD>
        <type>
          <answerType>fact.quantity</answerType>
        </type>
      </AQD>
      <fallback_question>Please tell me the second number!</
        fallback_question>
      <required>true</required>
      <useLG>>false</useLG>
    </ito>
    <ito name="op">
      <AQD>
        <type>
          <answerType>item(add,subtract,multiply,divide)</answerType>
        </type>
      </AQD>
      <fallback_question>Please tell me the operation!</fallback_question>
      <required>true</required>
      <useLG>>false</useLG>
    </ito>
  </itos>
  <action>
    <groovyAction>
      <returnAnswer>true</returnAnswer>
      <utteranceTemplate>%number1 %op %number2 is #res.</utteranceTemplate>
      <code><![CDATA[
Integer one=new Integer(frame.get("number1"));
Integer two=new Integer(frame.get("number2"));
Float res=0.0;
switch(frame.get("op")){
case "ADD": res=one+two; break;
case "SUBTRACT": res=one-two; break;
case "DIVIDE": res=one/two; break;
}
]]></code>
    </groovyAction>
  </action>
</task>

```

Appendix

```
    case "MULTIPLY": res=one*two; break;
    }
    executionResults.put("res",res.toString());
  ]]</code>
</groovyAction>
</action>
</task>
```

Listing A.2: Nadia Calculator

A.7 Expert Interviews

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@moberg.net

Expert Interview

Nadia - Natural Dialogue System

Dear researcher,

You have been selected as one of the experts to give a short assessment about the Nadia system. Nadia is based on the results of my PhD research and is used to verify the viability of the developed model. With the help of this software, it should be easier to create a specification of what a dialogue system should ask and what it can understand. This specification can then be automatically processed by the Nadia Dialogue Engine. You find the manual and software under <http://moberg.net/nadia>. Please phrase a short assessment about Nadia (e.g. what you think is particularly useful and new) and evaluate the given attributes. Thank you very much for your support!



I understand, that by filling in this document, I agree that my assessment may be published in Markus Berg's PhD thesis.

Name: Dr Ian O'Neill

Position / Affiliation: Lecturer, Computer Science, Queen's University Belfast

Industry Academia

What qualifies you as a natural language processing expert (e.g. projects/education/teaching)?

I teach Artificial Intelligence and Software Engineering at Queen's University Belfast. Since 1996 I have had a research interest in the areas of natural language processing and spoken dialogue systems.

1. Short assessment (free text):

Nadia deals very effectively with user-driven shifts in dialogue context. In this respect it makes a very valuable contribution to development of dialogue systems that capture something of the fluidity of natural conversation. The fact that it assists non-specialists in the process of dialogue design and implementation is an added bonus. The challenges presented by both the modelling and management of mixed-initiative dialogue are very considerable, even to the dialogue specialist. This thesis makes a commendable contribution to the field by helping system administrators give correct structure, and even correct language and register (English/German; formal/informal), to spoken user-system interaction that crosses from dialogue to sub-dialogue and back again.

2. How do you assess the following attributes?

- | | | | | | | | |
|---|-------------------|-----------------------|-----------------------|-----------------------|----------------------------------|-----------------------|-----------|
| a) Novelty of the approach: | state of the art | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | very new |
| b) Demand (do we need that?) | lower | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | higher |
| c) Ease of use (development): | hard | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | easy |
| d) Level of simplification of the development: | no simplification | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | high s. |
| e) How much does this model contribute to the creation of more natural dialogues? | not at all | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | very much |

Appendix

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@mmberg.net

Expert Interview

Nadia - Natural Dialogue System

Dear researcher,

You have been selected as one of the experts to give a short assessment about the Nadia system. Nadia is based on the results of my PhD research and is used to verify the viability of the developed model. With the help of this software, it should be easier to create a specification of what a dialogue system should ask and what it can understand. This specification can then be automatically processed by the Nadia Dialogue Engine. You find the manual and software under <http://mmberg.net/nadia>. Please phrase a short assessment about Nadia (e.g. what you think is particularly useful and new) and evaluate the given attributes. Thank you very much for your support!

I understand, that by filling in this document, I agree that my assessment may be published in Markus Berg's PhD thesis.

Name: Amy Isard

Position / Affiliation: Research Fellow, School of Informatics, University of Edinburgh

Industry Academia

What qualifies you as a natural language processing expert (e.g. projects/education/teaching)? I have a Masters degree in Artificial Intelligence, and have worked for 15 years on research projects which involve natural language processing. I have also lectured students and supervised MSc projects in natural language generation.

1. Short assessment (free text):

This work is a very interesting step towards software which will allow users to design a flexible dialogue system. It allows non-expert users to benefit from natural language generation technology to express the same content in multiple ways depending on the dialogue context, and to specify the structure of dialogues more fully than in existing freely-available systems. It allows mixed-initiative dialogue strategies, which together with the ability to vary the language used, will lead to more natural and varied dialogues. The software makes the process of designing dialogues in a new area much faster and simpler, and would be useful to many different application domains.

2. How do you assess the following attributes?

- | | | | | | | | |
|---|-------------------|-----------------------|-----------------------|-----------------------|----------------------------------|----------------------------------|-----------|
| a) Novelty of the approach: | state of the art | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | very new |
| b) Demand (do we need that?) | lower | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | higher |
| c) Ease of use (development): | hard | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | easy |
| d) Level of simplification of the development: | no simplification | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | high s. |
| e) How much does this model contribute to the creation of more natural dialogues? | not at all | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | very much |

Appendix

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@mberg.net

Expert Interview

Nadia - Natural Dialogue System

Dear researcher,

You have been selected as one of the experts to give a short assessment about the Nadia system. Nadia is based on the results of my PhD research and is used to verify the viability of the developed model. With the help of this software, it should be easier to create a specification of what a dialogue system should ask and what it can understand. This specification can then be automatically processed by the Nadia Dialogue Engine. You find the manual and software under <http://mberg.net/nadia>. Please phrase a short assessment about Nadia (e.g. what you think is particularly useful and new) and evaluate the given attributes. Thank you very much for your support!




I understand, that by filling in this document, I agree that my assessment may be published in Markus Berg's PhD thesis.

Name: Flavius Frasinca

Position / Affiliation: Assistant Professor/Erasmus University Rotterdam

Industry Academia

What qualifies you as a natural language processing expert (e.g. projects/education/teaching)?

I am currently running an NWO (Netherlands Organisation for Scientific Research) project, Financial Events Recognition in News for Algorithmic Trading (FERNAT), and participating as supervisor in another NWO project, Semantic Web Enhanced Product Search (SWEPS) and a Dutch Ministry of Economic Affairs (FES) project, COMMIT, where natural language processing plays a crucial role for understanding, personalizing, and recommending information. I also supervised numerous bachelor and master theses on application of natural language processing techniques in developing intelligent Web applications. 

1. Short assessment (free text):

The Nadia system proposes a dialogue model that describes the information demand and corresponding action that needs to be executed. One of the most innovative aspects of the dialogue model is its genericity and ability to specialize by means of configurations. For example, the granularity level of the dialogue strategies allow one to have more control on the dialogue structure than by using existing, state-of-the-art, systems.

Recently, a lot of work has been done on verbalizing ontologies as well as on ontology-based information extraction techniques that could help in question formulation and answering tasks described in NADIA (that go beyond keyword matching). Nevertheless, from an information-centric perspective, the proposed system brings adequate structures on the modeling of dialogues and their customization.

2. How do you assess the following attributes?

- | | | | |
|---|-------------------|--|-----------|
| a) Novelty of the approach: | state of the art | <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> | very new |
| b) Demand (do we need that?) | lower | <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> | higher |
| c) Ease of use (development): | hard | <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> | easy |
| d) Level of simplification of the development: | no simplification | <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> | high s. |
| e) How much does this model contribute to the creation of more natural dialogues? | not at all | <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> | very much |

Appendix

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@mmberg.net

Expert Interview

Nadia - Natural Dialogue System

Dear researcher,

You have been selected as one of the experts to give a short assessment about the Nadia system. Nadia is based on the results of my PhD research and is used to verify the viability of the developed model. With the help of this software, it should be easier to create a specification of what a dialogue system should ask and what it can understand. This specification can then be automatically processed by the Nadia Dialogue Engine. You find the manual and software under <http://mmberg.net/nadia>. Please phrase a short assessment about Nadia (e.g. what you think is particularly useful and new) and evaluate the given attributes. Thank you very much for your support!

I understand, that by filling in this document, I agree that my assessment may be published in Markus Berg's PhD thesis.

Name: Ashwin Ittoo

Position / Affiliation: Asst Prof, HEC Business School, Univ of Liege, Belgium

Industry Academia

What qualifies you as a natural language processing expert (e.g. projects/education/teaching)?

I did my PhD in this area, and I do research in NLP. I have substantial collaborations with industry on developing NLP applications (opinion mining).

1. Short assessment (free text):

In my opinion, the candidate, Mr. Berg, has made an interesting contribution to the area of Natural Language Generation and Understanding, and to Dialogue Systems. I feel that the NLP community is increasingly moving towards NLG, and in that respect, the Nadia system of Mr. Berg seems timely. There is currently a lack of "easy to use" dialogue system and Nadia tries to bridge this gap. Furthermore, the proposed tool, Nadia, presents several strenghts compared to more traditional tools like VoiceXML. I think that Nadia can serve as the foundation for more sophisticated dialogue-systems that are easy to use, and there is still long of work to be done in that area

2. How do you assess the following attributes?

- | | | | | | | | |
|---|-------------------|-----------------------|-----------------------|----------------------------------|----------------------------------|-----------------------|-----------|
| a) Novelty of the approach: | state of the art | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | very new |
| b) Demand (do we need that?) | lower | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | higher |
| c) Ease of use (development): | hard | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | easy |
| d) Level of simplification of the development: | no simplification | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | high s. |
| e) How much does this model contribute to the creation of more natural dialogues? | not at all | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | very much |

Appendix

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@mberg.net

Expert Interview

Nadia - Natural Dialogue System

Dear researcher,

You have been selected as one of the experts to give a short assessment about the Nadia system. Nadia is based on the results of my PhD research and is used to verify the viability of the developed model. With the help of this software, it should be easier to create a specification of what a dialogue system should ask and what it can understand. This specification can then be automatically processed by the Nadia Dialogue Engine. You find the manual and software under <http://mberg.net/nadia>. Please phrase a short assessment about Nadia (e.g. what you think is particularly useful and new) and evaluate the given attributes. Thank you very much for your support!

I understand, that by filling in this document, I agree that my assessment may be published in Markus Berg's PhD thesis.

Name: Jolanta Bachan

Position / Affiliation: Assistant professor, Adam Mickiewicz University, Poznań, Poland

Industry Academia

What qualifies you as a natural language processing expert (e.g. projects/education/teaching)?

I work in the speech technology projects: creation of the speech synthesis and the speech recognition systems. My PhD thesis was connected with creating a demo dialogue system.

1. Short assessment (free text):

The Nadia dialogue system is a well-designed system which allows the user to communicate with the system in a very natural way. The user can change the tasks/switch topics of the conversation and correct their answers. The interesting solution is to get online information from Wikipedia, Openweather and Google Maps for the dialogue.

2. How do you assess the following attributes?

- | | | | | | | | |
|---|-------------------|-----------------------|-----------------------|----------------------------------|-----------------------|----------------------------------|-----------|
| a) Novelty of the approach: | state of the art | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | very new |
| b) Demand (do we need that?) | lower | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | higher |
| c) Ease of use (development): | hard | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | easy |
| d) Level of simplification of the development: | no simplification | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | high s. |
| e) How much does this model contribute to the creation of more natural dialogues? | not at all | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | very much |

Appendix

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@mberg.net

Expert Interview

Nadia - Natural Dialogue System

Dear researcher,

You have been selected as one of the experts to give a short assessment about the Nadia system. Nadia is based on the results of my PhD research and is used to verify the viability of the developed model. With the help of this software, it should be easier to create a specification of what a dialogue system should ask and what it can understand. This specification can then be automatically processed by the Nadia Dialogue Engine. You find the manual and software under <http://mberg.net/nadia>. Please phrase a short assessment about Nadia (e.g. what you think is particularly useful and new) and evaluate the given attributes. Thank you very much for your support!

I understand, that by filling in this document, I agree that my assessment may be published in Markus Berg's PhD thesis.

Name: Sacher, Ronald

Position / Affiliation: IT-Projektleiter

Industry Academia

What qualifies you as a natural language processing expert (e.g. projects/education/teaching)?

Forschungsprojekte im Bereich Sprachverarbeitung: TravelConsult im Dialog und Virtuelle Rezeption

1. Short assessment (free text):

Unter der Voraussetzung, dass man mit dem Modell und der Software bereits vertraut ist, gilt:

- * Verringert die Einstiegshürde
- * Beschleunigt Entwicklung und ermöglicht Rapid Prototyping
- * Kapselt die Komponenten klar: erhöht Softwarequalität und Wiederverwendbarkeit

Die Aussicht, dass das Projekt Open Source wird, ist für Firmen sehr attraktiv.
Insgesamt ein absolut hilfreicher Evolutionsschritt in der Sprachverarbeitungswelt.

2. How do you assess the following attributes?

- | | | | | | | | |
|---|-------------------|-----------------------|-----------------------|----------------------------------|----------------------------------|----------------------------------|-----------|
| a) Novelty of the approach: | state of the art | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | very new |
| b) Demand (do we need that?) | lower | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | higher |
| c) Ease of use (development): | hard | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | easy |
| d) Level of simplification of the development: | no simplification | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | high s. |
| e) How much does this model contribute to the creation of more natural dialogues? | not at all | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | very much |

Appendix

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@mmborg.net

Expert Interview

Nadia - Natural Dialogue System

Dear researcher,

You have been selected as one of the experts to give a short assessment about the Nadia system. Nadia is based on the results of my PhD research and is used to verify the viability of the developed model. With the help of this software, it should be easier to create a specification of what a dialogue system should ask and what it can understand. This specification can then be automatically processed by the Nadia Dialogue Engine. You find the manual and software under <http://mmborg.net/nadia>. Please phrase a short assessment about Nadia (e.g. what you think is particularly useful and new) and evaluate the given attributes. Thank you very much for your support!

I understand, that by filling in this document, I agree that my assessment may be published in Markus Berg's PhD thesis.

Name: Stefan Kalkbrenner

Position / Affiliation: Principal IT-Consultant

Industry Academia

What qualifies you as a natural language processing expert (e.g. projects/education/teaching)?

Stefan Kalkbrenner has a degree in computer science. Starting as research associate in the field of speech technology in 2004, he became CEO of the Institute of Multimedia Engineering in 2006. Mr. Kalkbrenner has left the institute in 2012 and became an IT-Consultant at Namics (Deutschland) GmbH.

1. Short assessment (free text):

Setting up speech dialogue systems and creating natural dialogues are still complex challenges. Due to products like Apple's 'Siri' speech dialogue systems are already accepted in the market. Today there is a need to reduce the complexity of the development and for a decrease of the time to market.

Mr. Berg developed 'Nadia' within his PhD-Thesis as a contribution for the addressed problems. Nadia is set of components that deals with the creation of spoken dialogue systems and combines dialogue acts, concept-to-text language generation of system questions, answer types and natural language understanding modules.

Nadia allows students to generate dialogues without deep coding expertise. The reusability of components and interfaces is leads to fast development. Therefore a significant decrease of development time is possible.

2. How do you assess the following attributes?

a) Novelty of the approach:	state of the art	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	very new
b) Demand (do we need that?)	lower	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	higher
c) Ease of use (development):	hard	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	easy
d) Level of simplification of the development:	no simplification	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	high s.
e) How much does this model contribute to the creation of more natural dialogues?	not at all	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	very much

A.8 User Study

The following document has been presented to the participants¹. It contains the task description and general information about the study.

Markus M. Berg
University of Kiel / University of Wismar, Germany
mail@mberg.net

User Study

Nadia - Natural Dialogue System

Dear participant,

Nadia is a software to create and process a specification of a natural dialogue, i.e. a description of what the dialogue system can ask and what it is able to understand. Nadia is based on the results of my PhD research and is used to verify the viability of the developed dialogue model. Please read the manual and familiarise yourself with the Nadia system. Afterwards, you are asked to create a dialogue system with different tasks. Finally, please evaluate the development process and the resulting dialogue.

First, create the following dialogue system with either the help of the Java library or directly in XML.

1. Create a dialogue system that asks for the temperature in a given city. You may use the *Openweathermap API* (<http://openweathermap.org/api>) in connection with an *HTTPAction*.
2. Extend the dialogue with a new task that determines the shortest time between two cities (by car). You can use the *Google Directions API*:
<https://developers.google.com/maps/documentation/directions/>, e.g.
<http://maps.googleapis.com/maps/api/directions/xml?origin=Edinburgh&destination=Glasgow&sensor=false>
3. Add an open question for being able to switch between the tasks.
4. Add another task: Create a price information system for flights (attributes: *start city, destination city, start date, return date, number of persons*). You can simulate the back end with a *Groovy* script. Please use the *AQD* to describe and generate the system questions.

You should now have created a dialogue system that supports dialogues like these:

- *How may I help you?*
- *How long do I need from Edinburgh to Glasgow?*
- *59 Mins.*
- *How is the weather in Glasgow?*
- *18°C*
- *...*
- *How may I help you?*
- *I'd like to book a trip*
- *Where do you want to start?*
- *In Edinburgh*
- *And where do you want to go?*
- *How is the weather in Inverness?*
- *20°C. And what is your destination?*
- *Umm, how long do I need from Edinburgh to Inverness?*
- *2:56 hrs. And where do you want to go?*
- *Ok, to Inverness then.*
- *When do you want to depart?*
- *I'd like to travel from 01/08/2013 until 14/08/2013.*
- *And with how many persons?*
- *3*
- *The cheapest flight is 500 Euro.*

¹K. Brodkorb, H. Fanter, A. Niekamp, M. Sötebier, M. Thielbeer, T. Wendel

Appendix

Results of participant 1:

User Study

Nadia - Natural Dialogue System

Development:

Was it possible to realise the demands?	Some	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	All
How difficult was it in general to create the dialogue?	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• subdialogues	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• back end access	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• adaptive formulation	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• open question	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• mixed initiative	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• language understanding	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
How difficult was it to create the dialogue compared to Voice XML?	Easier	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Harder
How satisfied are you with the development process?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Completely

Dialogue:

How satisfied are you with the resulting dialogue?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Completely
Compare the resulting dialogue to state of the art systems! Is it...	Worse	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Better
How natural is the resulting dialogue?	Machine-like	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/>	Human-like

Results of participant 2:

User Study

Nadia - Natural Dialogue System

Development:

Was it possible to realise the demands?	Some	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	All
How difficult was it in general to create the dialogue?	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• subdialogues	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• back end access	Easy	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• adaptive formulation	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• open question	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• mixed initiative	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• language understanding	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
How difficult was it to create the dialogue compared to Voice XML?	Easier	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Harder
How satisfied are you with the development process?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Completely

Dialogue:

How satisfied are you with the resulting dialogue?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Completely
Compare the resulting dialogue to state of the art systems! Is it...	Worse	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Better
How natural is the resulting dialogue?	Machine-like	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Human-like

Appendix

Results of participant 3:

User Study

Nadia - Natural Dialogue System

Development:

Was it possible to realise the demands?	Some	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	All
How difficult was it in general to create the dialogue?	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• subdialogues	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• back end access	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• adaptive formulation	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• open question	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• mixed initiative	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• language understanding	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
How difficult was it to create the dialogue compared to Voice XML?	Easier	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Harder
How satisfied are you with the development process?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Completely

Dialogue:

How satisfied are you with the resulting dialogue?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Completely
Compare the resulting dialogue to state of the art systems! Is it...	Worse	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Better
How natural is the resulting dialogue?	Machine-like	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Human-like

Results of participant 4:

User Study

Nadia - Natural Dialogue System

Development:

Was it possible to realise the demands?	Some	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	All
How difficult was it in general to create the dialogue?	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• subdialogues	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• back end access	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• adaptive formulation	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• open question	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• mixed initiative	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• language understanding	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
How difficult was it to create the dialogue compared to Voice XML?	Easier	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Harder
How satisfied are you with the development process?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Completely

Dialogue:

How satisfied are you with the resulting dialogue?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Completely
Compare the resulting dialogue to state of the art systems! Is it...	Worse	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Better
How natural is the resulting dialogue?	Machine-like	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Human-like

Appendix

Results of participant 5:

User Study

Nadia - Natural Dialogue System

Development:

Was it possible to realise the demands?	Some	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	All
How difficult was it in general to create the dialogue?	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• subdialogues	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• back end access	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• adaptive formulation	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• open question	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• mixed initiative	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• language understanding	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
How difficult was it to create the dialogue compared to Voice XML?	Easier	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Harder
How satisfied are you with the development process?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Completely

Dialogue:

How satisfied are you with the resulting dialogue?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Completely
Compare the resulting dialogue to state of the art systems! Is it...	Worse	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Better
How natural is the resulting dialogue?	Machine-like	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/>	Human-like

Results of participant 6:

User Study

Nadia - Natural Dialogue System

Development:

Was it possible to realise the demands?	Some	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	All
How difficult was it in general to create the dialogue?	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• subdialogues	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• back end access	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• adaptive formulation	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• open question	Easy	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• mixed initiative	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
• language understanding	Easy	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Complicated
How difficult was it to create the dialogue compared to Voice XML?	Easier	<input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>	Harder
How satisfied are you with the development process?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Completely

Dialogue:

How satisfied are you with the resulting dialogue?	Not at all	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Completely
Compare the resulting dialogue to state of the art systems! Is it...	Worse	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>	Better
How natural is the resulting dialogue?	Machine-like	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>	Human-like

Bibliography

- Heinz Abels and Horst Stenger. *Gesellschaft lernen. Einführung in die Soziologie*. Leske und Budrich, 1986.
- J. Allen and M. Core. Draft of DAMSL: Dialog Act Markup in Several Layers, 1997. URL <http://www.cs.rochester.edu/research/speech/damsl/RevisedManual/>.
- J. L. Austin. *How to do things with words*. Harvard University Press, Cambridge, Mass., 1975.
- Jason Baldridge and Geert-Jan M. Kruijff. Multi-modal Combinatory Categorical Grammar. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics*, volume 1, pages 211–218, Stroudsburg, PA, USA, 2003. ISBN 1-333-56789-0.
- Linda Bell. *Linguistic Adaptations in Spoken Human-Computer Dialogues*. PhD thesis, KTH Stockholm, 2003.
- Markus Berg. Entwicklung eines Architekturkonzeptes für ein natürlichsprachliches, dialogbasiertes Raumsteuerungssystem. Master’s thesis, University of Wismar, Apr 2009.
- Markus Berg. Towards the Modelling of Backend Functionalities in Task-Oriented Dialogue Systems. In *5th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, Poznan (Poland), 11 2011. ISBN 978-83-932640-1-8.
- Markus Berg. Survey on Spoken Dialogue Systems: User Expectations regarding Style and Usability. In *XIV International PhD Workshop*, Wisla (Poland), 10 2012. ISBN 978-83-935427-0-3.
- Markus Berg. Natürlichsprachlichkeit in Dialogsystemen. *Informatik-Spektrum*, 36(4): 371–381, 2013. ISSN 0170-6012.
- Markus Berg and Antje Düsterhöft. Website Interaction with Text-based Natural Language Dialog Systems. In *7. Wismarer Wirtschaftsinformatiktage*, Wismar (Germany), 6 2010. ISBN 978-3-939159-84-1.
- Markus Berg, Nils Weber, Christoph Eigenstetter, and Antje Düsterhöft. Natürlichsprachliche Dialoge in Raumsteuerungssystemen. In *4. Kongress Multimediatechnik*, Wismar (Germany), 10 2009.

- Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Integration of Natural Language Dialogues into the Conceptual Model of Storyboard Design. In *Natural Language Processing and Information Systems, 15th International Conference on Applications of Natural Language to Information Systems*, volume 6177 of *Lecture Notes in Computer Science*, pages 196–203, Cardiff (UK), 7 2010a. Springer.
- Markus Berg, Petra Gröber, and Martina Weicht. User Study: Talking to Computers. In *3rd Workshop on inclusive eLearning*, London (UK), 9 2010b. ISBN 978-3-88120-811-6.
- Markus Berg, Bernhard Thalheim, and Antje Düsterhöft. Integration of Dialogue Patterns into the Conceptual Model of Storyboard Design. In *Advances in Conceptual Modeling - Applications and Challenges, Proceedings of ER Workshops*, volume 6413 of *Lecture Notes in Computer Science*, pages 160–169, Vancouver (Canada), 11 2010c. Springer.
- Markus Berg, Nils Weber, Gernot Ruscher, and Sebastian Bader. Maike: Mobile Assistenzsysteme für intelligente kooperierende Räume und Ensembles. In *5. Kongress Multimediatechnik*, Wismar (Germany), 10 2010d.
- Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Adaptation in Speech Dialogues – Possibilities to Make Human-Computer-Interaction More Natural. In *Fifth Baltic Conference "Human-Computer-Interaction"*, Riga (Latvia), 08 2011a. ISBN 978-3-86009-127-2.
- Markus Berg, Bernhard Thalheim, and Antje Düsterhöft. Dialog Acts from the Processing Perspective in Task Oriented Dialog Systems. In *Proceedings of the 15th Workshop on the Semantics and Pragmatics of Dialogue (SemDial 2011)*, pages 176–177, Los Angeles (USA), 09 2011b.
- Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Towards Interrogative Types in Task-oriented Dialogue Systems. In *17th International Conference on Applications of Natural Language Processing to Information Systems*, volume 7337 of *Lecture Notes in Computer Science*, Groningen (The Netherlands), 07 2012a. Springer. ISBN 978-3-642-31177-2.
- Markus Berg, Antje Düsterhöft, and Bernhard Thalheim. Query and Answer Forms for Sophisticated Database Interfaces. In *22nd European Japanese Conference on Information Modelling and Knowledge Bases*, Prague (Czech Republic), 07 2012b. ISBN 978-1-61499-176-2.
- Markus Berg, Antje Düsterhöft, Nils Weber, and Christoph Eigenstetter. Wirtschaftlicher Mehrwert durch den Einsatz von Sprachtechnologien. In *Fachkongress Social Business – Tagungsband 2013*, pages 24–32. eBusiness-Lotse NordOst, 2013a. ISBN 978-3-00-043454-9.
- Markus M. Berg, Amy Isard, and Johanna D. Moore. An OpenCCG-Based Approach to Question Generation from Concepts. In Elisabeth Métais, Farid Meziane, Mohamad

- Saraee, Vijayan Sugumaran, and Sunil Vadera, editors, *Natural Language Processing and Information Systems*, volume 7934 of *Lecture Notes in Computer Science*, pages 38–52, Salford (UK), 2013b. Springer Berlin Heidelberg. ISBN 978-3-642-38823-1.
- Anna Biselli, Patrick Reipschläger, Markus Berg, and Antje Düsterhöft. Automatische Übersetzung von der Deutschen Gebärdensprache in die deutsche Lautsprache. In *6. Kongress Multimedialechnik*, Wismar (Germany), 10 2011.
- B. S. Bloom et al. *Taxonomy of educational objectives. The classification of educational goals. Handbook 1: Cognitive domain*. Longmans Green, 1956.
- Johan Bos, Ewan Klein, Oliver Lemon, and Tetsushi Oka. DIPPER: Description and formalisation of an information-state update dialogue system architecture. In *4th SIGdial Workshop on Discourse and Dialogue*, 4th SIGdial Workshop on Discourse and Dialogue, pages 115–124, 2003.
- Kristy Elizabeth Boyer and Paul Piwek, editors. *Proceedings of QG2010: The Third Workshop on Question Generation*, Pittsburgh, 2010.
- Kristy Elizabeth Boyer et al. An Empirically Derived Question Taxonomy for Task-Oriented Tutorial Dialogue. In *Proceedings of the Second Workshop on Question Generation*, Proceedings of the Second Workshop on Question Generation, pages 9–16, Brighton, U.K., 2009.
- Susan Brennan. Lexical Entrainment In Spontaneous Dialog. In *International Symposium on Spoken Dialog*, 1996.
- Susan E. Brennan and Justina O. Ohaeri. Effects of message style on users’ attributions toward agents. In *CHI '94 Conference Companion on Human Factors in Computing Systems*, Conference companion on Human factors in computing systems, pages 281–282, New York, NY, USA, 1994. ACM. ISBN 0-89791-651-4.
- Björn Bringert. *Programming Language Techniques for Natural Language Applications*. PhD thesis, University of Gothenburg, 2008.
- P. Brown, S. C. Levinson, and J. J. Gumperz. *Politeness: Some Universals in Language Usage*. Studies in Interactional Sociolinguistics. Cambridge University Press, 1987. ISBN 9780521313551.
- Penelope Brown and Stephen C. Levinson. *Politeness: Some Universals in Language Usage (Studies in Interactional Sociolinguistics)*. Cambridge University Press, 1987.
- Harry Bunt. Dimensions in Dialogue Act Annotation. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006), Genova, Italy, 2006.
- Harry Bunt. DIT++ Taxonomy of Dialogue Acts, Aug 2010. URL <http://dit.uvt.nl/>.

- Harry Bunt et al. Towards an ISO Standard for Dialogue Act Annotation. In Nicoletta Calzolari et al., editors, *7th International Conference on Language Resources and Evaluation*, Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC 2010), Valletta, Malta, 2010. European Language Resources Association (ELRA).
- Jean Carletta, Amy Isard, et al. HCRC Dialogue Structure Coding Manual. Technical report, University of Edinburgh, 1996.
- Nancy Chinchor. MUC-7 Named Entity Task Definition, 1997. URL http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ne_task.html.
- Ronald A. Cole. Tools for Research and Education in Speech Science. In *Proceedings of the International Conference of Phonetic Sciences*, pages 1277–1280, 1999.
- Mark G. Core and James F. Allen. Coding Dialogs with the DAMSL Annotation Scheme, 1997.
- Caroline Cullen, Ian O’Neill, and Philip Hanna. Flexible Natural Language Generation in Multiple Contexts. In Zygmunt Vetulani and Hans Uszkoreit, editors, *Human Language Technology. Challenges of the Information Society*, pages 142–153, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04234-8.
- Kerstin Dautenhahn, Sarah Woods, Christina Kaouri, Michael L. Walters, Kheng Lee Koay, and Iain Werry. What is a robot companion - friend, assistant or butler. In *International Conference on Intelligent Robots and Systems*, pages 1488–1493, 2005.
- Matthias Denecke. An integrated development environment for spoken dialogue systems. In *Proceedings of the COLING-2000 Workshop on Using Toolsets and Architectures To Build NLP Systems*, pages 51–60, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.
- Matthias Denecke. Rapid prototyping for spoken dialogue systems. In *Proceedings of the 19th International Conference on Computational linguistics*, volume 1, pages 1–7, Stroudsburg, PA, USA, 2002a. Association for Computational Linguistics.
- Matthias Denecke. *Generische Interaktionsmuster für aufgabenorientierte Dialogsysteme*. PhD thesis, Karlsruhe Institute of Technology, 2002b.
- Hans Dybkjær and Laila Dybkjær. DialogDesigner – A Tool for Rapid System Design and Evaluation. In *6th SIGdial Workshop on Discourse and Dialogue*, pages 227–231, 2005.
- Jens Edlund, Mattias Heldner, and Joakim Gustafson. Two faces of spoken dialogue systems. In *Interspeech 2006 - ICSLP Satellite Workshop Dialogue on Dialogues*, 2006.
- Jens Edlund, Joakim Gustafson, Mattias Heldner, and Anna Hjalmarsson. Towards human-like spoken dialogue systems. *Speech Commun.*, 50(8-9):630–645, Aug 2008. ISSN 01676393.

- Hans Fanter. Analyse von Techniken und Konzeption einer Anwendung zur Interpretation von natürlichsprachlichen Äußerungen innerhalb eines Dialogsystems. Master's thesis, University of Wismar, 2014. To appear.
- Mark Fishel. Dialogue Act Recognition Techniques, 2006.
- Mark Fishel. Machine Learning Techniques in Dialogue Act Recognition. *Estonian Papers in Applied Linguistics*, 3:117–134, 2007.
- Terrence Fong, Charles Thorpe, and Charles Baur. Collaboration, Dialogue, and Human-Robot Interaction. In *10th International Symposium of Robotics Research*, 2002.
- James R. Glass and Eugene Weinstein. Speechbuilder: Facilitating Spoken Dialogue System Development. In *Proc. of the 7th European Conference on Speech Communications and Technology*, INTERSPEECH, pages 1335–1338, 2001.
- Art Graesser, Vasile Rus, and Zhiqiang Cai. Question classification schemes. In *Proceedings of the Workshop on the Question Generation Shared Task and Evaluation Challenge*, pages 8–9, 2008.
- H. P. Grice. *Studies in the way of words*. Harvard University Press, 1989.
- Petra Gröber, Martina Weicht, and Markus Berg. Inclusive eLearning - Special Needs and Special Solutions? In *3rd Workshop on inclusive eLearning*, London (UK), 9 2010.
- Swati Gupta, Marilyn Walker, and Daniela Romano. Generating Politeness in Task Based Interaction: An Evaluation of the Effect of Linguistic Form and Culture. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, Proceedings of the Eleventh European Workshop on Natural Language Generation, pages 57–64, Saarbrücken, Germany, Jun 2007. DFKI GmbH. Document D-07-01.
- Jörg Hagemann and Rolf Eckard. *Text- und Gesprächslinguistik*, chapter Die Bedeutung der Sprechakttheorie für die Gesprächsforschung, pages 885–895. de Gruyter, 2001.
- Julia Hirschberg and Diane Litman. Empirical studies on the disambiguation of cue phrases. *Computational Linguistics*, 19:501–530, 1993.
- L. Hirschman and R. Gaizauskas. Natural language question answering: the view from here. *Nat. Lang. Eng.*, 7(4):275–300, 2001.
- Anna Hjalmarsson and Jens Edlund. Human-Likeness in Utterance Generation: Effects of Variability. In *Perception in Multimodal Dialogue Systems*, Proceedings of the 4th IEEE tutorial and research workshop on Perception and Interactive Technologies for Speech-Based Systems: Perception in Multimodal Dialogue Systems, pages 252–255, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-69368-0.
- Julia Hockenmaier and Mark Steedman. CCGbank: User's Manual. Technical report, University of Pennsylvania, May 2005.

- Julia Hockenmaier and Mark Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Comput. Linguist.*, 33(3): 355–396, Sep 2007.
- Wolfgang Hoepfner. Review of "Generating natural language under pragmatic constraints" by Edward H. Hovy. *Computational Linguistics*, 16(3):186–189, 1990.
- T. Holtgraves. *Language As Social Action: Social Psychology and Language Use*. Psychology Press, 2001. ISBN 9780805831405.
- Eduard Hovy, Laurie Gerber, et al. Question Answering in Webclopedia. In *Proceedings of the Ninth Text REtrieval Conference*, In Proceedings of the Ninth Text Retrieval Conference, pages 655–664, 2000.
- Eduard H Hovy. *Generating natural language under pragmatic constraints*. PhD thesis, Yale University, New Haven, CT, USA, 1987. AAI8729079.
- Eduard H. Hovy. Pragmatics and natural language generation. *Artif. Intell.*, 43(2): 153–197, May 1990. ISSN 00043702.
- Amy Isard, Carsten Brockmann, and Jon Oberlander. Individuality and alignment in generated dialogues. In *Proceedings of the Fourth International Natural Language Generation Conference*, Proceedings of the Fourth International Natural Language Generation Conference, pages 25–32, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. ISBN 1-932432-72-8.
- Michael Jaeckel. Interaktion. Soziologische Anmerkungen zu einem Begriff. *Rundfunk und Fernsehen*, 4:463–476, 1995.
- Kristiina Jokinen and Michael F. McTear. *Spoken Dialogue Systems*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2009.
- Kristiina Jokinen and Graham Wilcock. *Adaptivity and Response Generation in a Spoken Dialogue System*, 2007.
- Markus de Jong, Mariët Theune, and Dennis Hof. Politeness and Alignment in Dialogues with a Virtual Guide. In *Seventh International Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 207–214, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-0-9.
- Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0130950696.
- Gisela Klann Delius. *Text- und Gesprächslinguistik*, chapter Bedingungen und Möglichkeiten verbaler Kommunikation, pages 1115–1121. de Gruyter, 2001.
- Geert-Jan M. Kruijff, Michael White, and Cem Bozşahin. *Specifying Grammars for OpenCCG: A Rough Guide*, 2005.

- Ivana Kruijff Korbayová, Ciprian Gerstenberger Olga Kukina, and Jan Schehl. Generation of output style variation in the SAMMIE dialogue system. In *Proceedings of the Fifth International Natural Language Generation Conference*, pages 129–137, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.
- Staffan Larsson. *Issue-based Dialogue Management*. PhD thesis, Göteborg University, Göteborg, Sweden, 2002.
- Staffan Larsson, Alexander Berman, Johan Bos, Leif Grönqvist, Peter Ljunglöf, and David Traum. TrindiKit 2.0 Manual. Technical report, Gothenburg University, 2000.
- Geoffrey N. Leech. *Principles of Pragmatics*. Longman, 1983.
- Oliver Lemon and Xingkun Liu. DUDE: a Dialogue and Understanding Development Environment, Mapping Business Process Models to Information State Update Dialogue Systems. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Posters & Demonstrations*, pages 99–102, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.
- Oliver Lemon, Xingkun Liu, and Helen Hastie. Build your own spoken dialogue systems: automatically generating ISU dialogue systems from business user resources. In *22nd International Conference on Computational Linguistics: Demonstration Papers*, pages 161–164, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.
- Qin En Looi and Swee Lan See. Applying Politeness Maxims in Social Robotics Polite Dialogue. In *International Conference on Human-Robot Interaction*, pages 189–190, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1063-5.
- Hui Ma, Klaus-Dieter Schewe, and Bernhard Thalheim. Context Analysis: Toward Pragmatics of Web Information Systems Design. In *Fifth Asia-Pacific Conference on Conceptual Modelling*, volume 79. Australian Computer Society, 2008.
- François Mairesse. *Learning to Adapt in Dialogue Systems: Data-driven Models for Personality Recognition and Generation*. PhD thesis, University of Sheffield, Feb 2008.
- H. Mangold. *Sprachliche Mensch-Maschine-Kommunikation*, chapter Im Dialog mit Maschinen. Oldenbourg, 1992.
- Christopher D. Manning and Dan Klein. Optimization, Maxent Models, and Conditional Estimation without Magic. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, HLT-NAACL, 2003.
- Michael F. McTear. Software To Support Research And Development Of Spoken Dialogue Systems. In *Proceedings of Eurospeech '99*, pages 339–342, 1999a.

- Michael F. McTear. Using the CSLU toolkit for practicals in spoken dialogue technology, 1999b.
- Michael F. McTear. *Spoken Dialogue Technology: Toward the Conversational User Interface*. Springer, 2004.
- Houghton Mifflin. *The American Heritage Dictionary of the English Language*. Houghton Mifflin Company, 4th edition, 2009.
- Dan I. Moldovan et al. LASSO: A Tool for Surfing the Answer Net. In *TREC-8 Proceedings*, Text Retrieval Conference, 1999.
- Knud Möller. Lifecycle models of data-centric systems and domains. *Semantic Web Journal*, 4:67–88, 2013.
- Clifford Nass and Kwan Min Lee. Does computer-generated speech manifest personality? An experimental test of similarity-attraction. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 329–336, New York, NY, USA, 2000. ACM. ISBN 1-58113-216-6.
- Scott Nowson. *The Language of Weblogs: A study of genre and individual differences*. PhD thesis, University of Edinburgh, 2006.
- A. M. Olney, A. C. Graesser, and N. K. Person. Question Generation from Concept Maps. *Dialogue and Discourse*, 3(2):75–99, 2012.
- Ian M. O’Neill, Philip Hanna, Xingkun Liu, and Michael F. McTear. The Queen’s Communicator: An Object-Oriented Dialogue Manager. In *Proceedings of Eurospeech*, 2003.
- Shiyan Ou, Constantin Orasan, Dalila Mekhaldi, and Laura Hasler. Automatic Question Pattern Generation for Ontology-based Question Answering. In *Proceedings of the Twenty-First International FLAIRS Conference*, The Florida AI Research Society Conference, pages 183–188, 2008.
- Andreas Papasalouros, Konstantinos Kanaris, and Konstantinos Kotis. Automatic generation of multiple-choice questions from domain ontologies. In *IADIS International Conference e-Learning*, 2008.
- Norbert Pflieger. *Context-based multimodal interpretation: an integrated approach to multimodal fusion and discourse processing*. PhD thesis, Universität des Saarlandes, 2008.
- Silvia Quarteroni. Personalized, interactive question answering on the Web. In *Proceedings of the Workshop on Knowledge and Reasoning for Answering Questions*, pages 33–40, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. ISBN 978-1-905593-53-8.

- HoIger Quast, Tobias Scheideck, Petra Geutner, and Andreas Korthauer. RoBoDiMa: a dialog object based natural language speech dialog system. In *Proceedings of Automatic Speech Recognition and Understanding*, IEEE Workshop on Automatic Speech Recognition and Understanding, 2003. Key: quast03.
- Bhavani Raskutti and Ingrid Zukerman. Generating Queries and Replies during Information-seeking Interactions. *Int. J. Hum.-Comput. Stud.*, 47(6):689–734, Dec 1997. ISSN 10715819.
- Ehud Reiter and Robert Dale. *Building natural language generation systems*. Cambridge university press, 2000.
- Vasile Rus and James Lester, editors. *AIED 2009 Workshops Proceedings: The 2nd Workshop on Question Generation*, Brighton, 2009.
- Bernhard Schäfers. *Grundbegriffe der Soziologie*. Verlag für Sozialwissenschaften, Wiesbaden, 2006.
- E. A. Schegloff. *Sequence Organization in Interaction: A Primer in Conversation Analysis*, volume 1 of *Sequence Organization in Interaction: A Primer in Conversation Analysis*. Cambridge University Press, 2007. ISBN 9780521532792.
- Klaus-Dieter Schewe and Bernhard Thalheim. Conceptual modelling of web information systems. *Data & Knowledge Engineering*, 54(2):147–188, 2005. ISSN 0169023X.
- J. Searle. A taxonomy of illocutionary acts. In K. Gunderson, editor, *Language, Mind, and Knowledge*, pages 344–369. University of Minnesota Press, 1975.
- J. R. Searle. *Expression and Meaning: Studies in the Theory of Speech Acts*. Cambridge University Press, 1979.
- John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.
- John R. Searle. *(On) Searle on conversation*. J. Benjamins Pub. Co., 1992.
- Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.
- Maren Sötebier. Entwicklung einer webbasierten dynamisch generierten Oberfläche für eine Entwicklungsumgebung zur Erstellung von Sprachdialogen. Master’s thesis, University of Wismar, Dec 2013.
- Rohini Srihari and Wei Li. Information Extraction Supported Question Answering. In *Proceedings of the Eighth Text Retrieval Conference*, pages 185–196, 1999.
- Mark Steedman and Jason Baldridge. *Combinatory Categorical Grammar*, chapter Non-Transformational Syntax, pages 181–224. Wiley-Blackwell, 2011. ISBN 9781444395037.

- Amanda J. Stent. *Dialogue Systems as Conversational Partners: Applying Conversation Acts Theory to Natural Language Generation for Task-Oriented Mixed-Initiative Spoken Dialogue*. PhD thesis, University of Rochester, 2001.
- Martin Stokhof and Jeroen Groenendijk. *Handbook of Logic and Language*, chapter 19, page 21. Elsevier Science B.V., 1994.
- Andreas Stolcke et al. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26:339–373, 2000.
- Marc Thielbeer. Anbindung eines Instant Messengers an das Sprachdialogsystem NADIA. Technical report, University of Wismar, Oct 2013.
- David Traum and Staffan Larsson. *Current and New Directions in Discourse and Dialogue*, volume 22, chapter The Information State Approach to Dialogue Management, pages 325–353. Kluwer Academic Publishers, 2003.
- Markku Turunen and Jaakko Hakulinen. Jaspis - A Framework for Multilingual Adaptive Speech Applications. In *Sixth International Conference on Spoken Language Processing*, pages 719–722, 2000.
- Zeno Vendler. Verbs and Times. *The Philosophical Review*, 66:143–160, 1957.
- H. G. Vester. *Kompendium der Soziologie I: Grundbegriffe*. Kompendium der Soziologie. VS Verlag für Sozialwissenschaften, 2008. ISBN 9783531158051.
- Friedemann Schulz von Thun. *Miteinander reden: Störungen und Klärungen*. Rowohlt, Reinbek bei Hamburg, 1983.
- Ellen M. Voorhees. Overview of the TREC 2003 Question Answering Track. In *The Twelfth Text Retrieval Conference*, The Twelfth Text Retrieval Conference, 2003.
- Nick Webb. *Cue-based dialogue act classification*. PhD thesis, Department of Computer Science, University of Sheffield, Sheffield, England, 2010.
- Nils Weber, Christoph Eigenstetter, Markus Berg, and Antje Düsterhöft. Ontologiespeicherung in Datenbanken im Kontext natürlichsprachlicher Dialogsysteme. In *21. Grundlagenworkshop von Datenbanken*, Rostock (Germany), 6 2009.
- Eugene Weinstein. SPEECHBUILDER: Facilitating Spoken Dialogue System Development. Master’s thesis, Massachusetts Institute of Technology, 2000.
- Joseph Weizenbaum. ELIZA - A computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, Jan 1966. ISSN 00010782.
- Tom Wendel and Marc Thielbeer. Entwicklung eines Sprachdialoges mit NADIA. Technical report, University of Wismar, Nov 2013.
- T. Winograd. *Understanding Natural Language*. Academic Press, New York, 1972.

List of Figures

2.1	Cost-benefit scale (Leech, 1983)	31
2.2	Indirectness scale (Leech, 1983)	31
3.1	Components of a dialogue system	39
3.2	Finite state dialogue (Jokinen and McTear, 2009)	41
3.3	Frame	42
3.4	Action scheme (Jurafsky and Martin, 2000)	43
3.5	Taxonomy of general purpose functions in DIT++ Bunt et al. (2010)	53
5.1	Wizard of Oz experiment	74
5.2	Results of the WOZ experiment	76
6.1	Survey questions	81
6.2	Have you ever used a SDS?	82
6.3	What do you think of open questions? (Questions without predefined answer like “How may I help you?”)	83
6.4	If you want to switch the light on, what do you say?	86
6.5	Which dialogue do you think is more pleasant?	87
6.6	You want to know how the weather will be. Which dialogue system would you use?	88
8.1	Back-end-oriented dialogue acts	97
8.2	Back-end-oriented dialogue acts and related functions	97
8.3	CRUD	99
8.4	Dialogue system	110
9.1	Answer Types	118
9.2	Abstract Question Description	124
10.1	NLG system architecture (Reiter and Dale, 2000)	128
10.2	A message about the monthly temperature (Reiter and Dale, 2000)	128
10.3	Abstract syntactic structure for ‘The month had some rainy days’ (Reiter and Dale, 2000)	129
10.4	NLG system scheme	134
10.5	Simplified parse for ‘When do you want to go?’	136
10.6	Logical form for ‘When do you want to go?’	137

10.7	Logical form for ‘Could you please tell me when you want to leave?’ . . .	137
10.8	Extract from ontology	144
10.9	Question generation as part of the whole dialogue system	145
10.10	Question generation with reference to the AQD	146
10.11	Survey	149
11.1	Information Transfer Object	156
11.2	Extract of a dialogue example	159
11.3	Extract of the dialogue model as schema definition	160
12.1	Schematic overview of the Natural Dialogue System	164
12.2	Web Client	165
12.3	Client-Server-Communication	165
12.4	Response log: initialisation of a new dialogue instance	166
12.5	Response log: retrieving the next system question	166
12.6	Preprocessing (Fanter, 2014)	171
12.7	Semantic Role Labelling: arguments for <i>travel</i> (Fanter, 2014)	171
12.8	Dialogue Development Environment	177
13.1	Visualisation of the light bulb tasks	181
14.1	NADIA expert survey: mean scores	190
14.2	NADIA user study mean scores	192
14.3	NADIA user study mean scores: development aspects in detail	193

List of Tables

8.1	Dialogue Acts vs CRUD	100
8.2	Cue words	107
8.3	Cue verbs	107
8.4	Potential features for recognising system-oriented dialogue acts	108
10.1	Lexical aspects	140
10.2	Dialogues sorted by politeness scores	149
14.1	Voice XML vs. Nadia	188
A.1	Politeness scores for fact questions	205
A.2	Politeness scores for decision questions	206