Chapter 2
Background

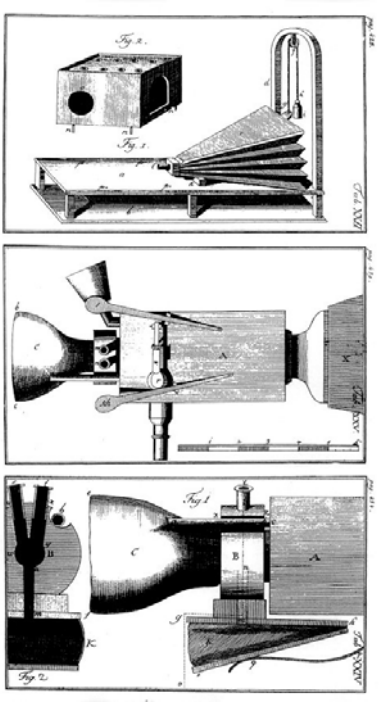

Fig. 2.1 Wolfgang von Kempelen's speaking machine, drawing from *Mechanism of Human Speech* (1791)

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

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

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

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

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

2.1 Human-Computer Interaction

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

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

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

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

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

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

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

2.2 Dialogue Strategy Development

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

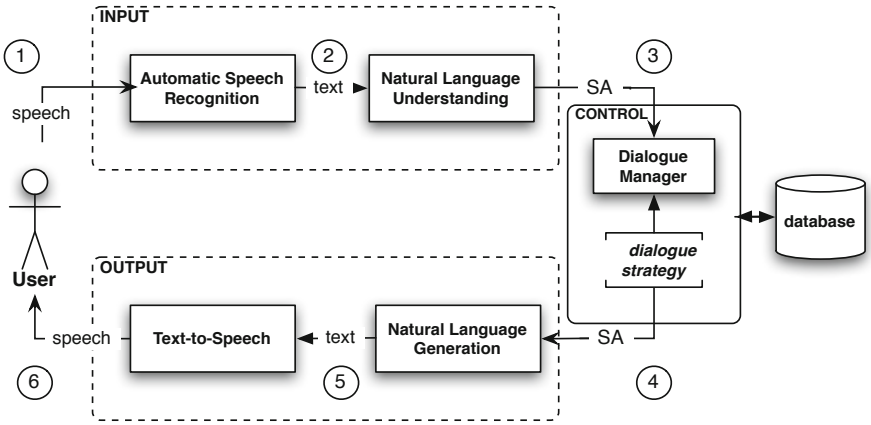

Fig. 2.2 Dialogue system architecture and processing pipeline

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

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

2.2.1 Conventional Development Lifecycle

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

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

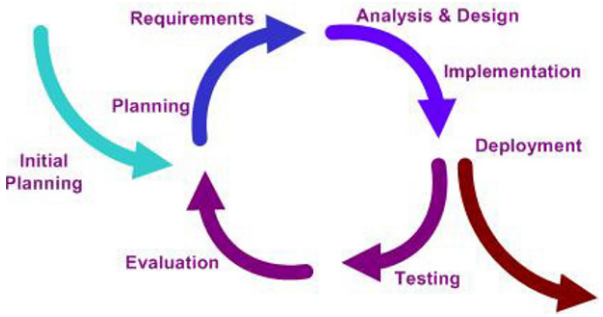

Fig. 2.3 Iterative development cycle used to develop dialogue systems

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

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

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

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

2.2.2 Evaluation and Strategy Quality Control

2.2.2.1 Quality Control in Industry

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

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

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

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

2.2.2.2 Evaluation Practises in Academia

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

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

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

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

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

2.2.2.3 The PARADISE Evaluation Framework

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

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

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

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

2.2.2.4 Strategy Re-Implementation

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

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

2.2.3 Strategy Implementation

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

2.2.3.1 Implementation Practises in Industry

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

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

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

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

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

Table 2.2 Frameworks conventionally used for dialogue strategy development

2.2.3.2 Implementation Practises in Academia

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

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

2.2.4 Challenges for Strategy Development

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

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

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

Second, good strategies should be *context sensitive* (Zue, 2007). Most current strategies make use of hand-coded thresholds in order to react to changes in the context. For example, thresholds are used to make strategies sensitive to the number

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

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

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

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

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

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

2.3 Literature review: Learning Dialogue Strategies

2.3.1 Machine Learning Paradigms

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

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

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

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

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

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

For dialogue strategy learning the simulated environment can include the (simulated) user, channel noise, the back-end database and other components of the dialogue system, such as ASR, NLU, and TTS. At each point in time t, the agent performs an action a_t and the environment generates an observation o_t and an instantaneous cost c_t (here also called "rewards"), according to some (usually unknown) dynamics. The goal is then to discover a policy for selecting actions that minimises

some measure of a long-term cost and maximises the expected cumulative utility (also known as 'final reward'). This is usually done by trial-and-error search methods. A detailed introduction to RL is given in Section 3.2.

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

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

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

2.3.2 Supervised Learning for Dialogue Strategies

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

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

A different approach to "human assisted design" is introduced by Okamoto et al (2001). They use a Wizard-of-Oz (WOZ) study (see Section 3.3.1) for data collection, where the human wizard is free to choose from a predefined set of utterances. They assume that learning an "average" strategy of decisions taken by the human wizard will result in a good dialogue strategy. Again, the wizard is not an expert in how a machine should ideally react in a certain situation. In general, human decisions often serve as "gold standard" for other tasks such as automatic summarisation (Teufel and van Halteren, 2004), or automatic annotation. However, human behaviour cannot be viewed as gold standard in the context of dialogue strategy design,

as Human-Machine Interaction is fundamentally different to Human-Human Interaction (as argued in Section 2.1). We will provide some examples of sub-optimal wizard behaviour in Chapter 6.4. In addition, (Okamoto et al, 2001) report that SL methods applied to WOZ data suffer from data-sparsity. (Note that WOZ studies are costly and therefore usually result in limited amounts of data.)

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

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

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

2.3.3 Dialogue as Decision Making under Uncertainty

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

$$A = \text{argmax}\, EU(a|o) = \underset{a}{\text{argmax}} \sum_{s} P(S = s|o) \times utility(a,s); \tag{2.2}$$

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

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

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

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

2.3.4 Reinforcement Learning for Dialogue Strategies

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

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

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

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

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

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

The work of Henderson et al (2005, 2008) addressed a much more complex problem using RL, which is represented by 10^{386} states and 74 actions (resulting in possible $74^{10^{386}}$ policies). They therefore learn from a much larger data set of 2331 dialogues and over 103000 turns, taken from the COMMUNICATOR corpus (Walker et al, 2002b). However, they find that state frequencies in the data set follow a Zipfian (i.e., long-tailed) distribution, with 61% of the system turns having states that only occurred once in the data. Henderson et al (2008) address the data sparsity problem using two different techniques. First, linear function approximation is applied. Linear function approximation learns linear estimates between state-action pairs and expected reward values (Sutton and Barto, 1998), and thus generalises to states which are not in the training data (also see Section 3.2.2.3). In addition,

Henderson et al (2008) apply a novel hybrid approach combining Supervised and Reinforcement Learning. SL is used to model which action to take in portions of the state space where data is not sufficient. RL is used to choose between the remaining states. The results show that the hybrid RL/SL policy outperforms the strategies which are based on RL or SL only. The SL policy is a very challenging baseline as it mimics a 'multi-version' system of 8 well-engineered COMMUNICATOR systems (Walker et al, 2002b). Multi-version systems are known to remove errors made by any one system that are not shared by most of the other systems.[8] Still, RL is able to improve upon SL by *exploring* different actions and evaluating how good they are in the long run (i.e. with respect to all subsequent actions which are likely to follow), rather than just choosing the locally best action according to the 8 systems. This result also illustrates that hand-coded strategies are very unlikely to be globally optimal: even the very "essence" of 8 expert systems manually designed by a human developer, is still sub-optimal. RL-based strategies, in contrast, are provably optimal. However, the exploration of new strategies as done by (Henderson et al, 2005, 2008) is very limited as they learn from a fixed data set. In addition, this approach relies on a substantial amount of initial training data, where annotated dialogue data is still scarce (Lemon and Pietquin, 2007).

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

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

2.4 Summary

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

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

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

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

http://www.springer.com/978-3-642-24941-9

Reinforcement Learning for Adaptive Dialogue Systems
A Data-driven Methodology for Dialogue Management
and Natural Language Generation
Rieser, V.; Lemon, O.
2011, XVI, 256 p., Hardcover
ISBN: 978-3-642-24941-9